PARTS OF THE ORIGINAL DOCUMENT

ARE OF POOR REPRODUCTIVE QUALITY

CERTAINES PARTIES DU DOCUMENT ORIGINAL

SONT DE PIÈTRE QUALITÉ DE REPRODUCTION

23369

Department of Mechanical Engineering

THE UNIVERSITY OF
BRITISH COLUMBIA
UBC Vancouver B.C. V6T 1W5 Canada

IGM86-201-142
CNRC 23369

Computer Aided Manufacturing and Robotics Laboratory

DEVELOPMENT OF AN AUTOMATIC PROGRAMMING SYSTEM FOR A WELDING ROBOT - PHASE 2: TECHNIQUES AND SOFTWARE FOR INTERFERENCE DETECTION AND PATH PLANNING

by

R.O. Buchal

D.B. Cherchas

J.P. Duncan

F. Sassani

February 1986

EXECUTIVE SUMMARY

The work described in this report is the second phase of a major project in automatic programming of a welding robot. The goals of the project are to develop, evaluate and select an optimum technique for detection of interference between objects in the workstation, to develop a path planning technique recognizing inverse kinematics and interference constraints and to demonstrate the complete package on an advanced graphics computer terminal.

The major issues covered by the report are the following: description of the workstation, development of kinematics, determination of a feasible welding trajectory that avoids constraints, interference detection methods, path planning, computer graphics methods, and development of an integrated software package.

1. Description of the Workstation

The workstation is specified by an external CAD system. The necessary data are divided into two parts:

(i) Geometric model - This is a representation of the topology and geometry of all solid objects in the workstation. This model is necessary for graphical depiction and interference detection. A model has been chosen that represents objects as clusters of convex polyhedra.

(ii) CAD seam description - This is a general, concise CAD description of the workpiece seams, including digitized points, seam orientation and workpiece welding parameters.

2. Kinematics

Fundamental kinematics theory is reviewed, and the kinematic relationships of a welding workstation are developed. The correct robot location is found as a function of the workpiece position and orientation, and the seam geometry.

3. Path feasibility

Path feasibility depends on the existence of solutions that do not violate joint limits or interference constraints. An explicit inverse kinematics solution for the PUMA robot is used. This solution can be applied to many robots of similar configuration. It is shown that five degrees of freedom are sufficient for welding, and that a sixth degree of freedom provides redundancy in the form of arbitrary rotations about the torch axis.

4. Interference detection

Four alternative interference detection methods are developed and evaluated. The method chosen quickly detects interference between convex ployhedra by searching for separating planes. These separating planes are also used for graphical hidden surface removal. If desired, exact separating distances can be found.

5. Path planning

Feasible, interference free paths must be found for each weld, as well as for motion between welds. To find a feasible welding trajectory, an arbitrary starting torch rotation and solution configuration are chosen. The inverse solution is then found for each point along the seam, starting with the first point. If the solution violates a joint limit or causes interference, a rotation is applied about the torch axis and the solution is recalculated. This is repeated until the solution is feasible, or until a limit is reached. If no feasible solution can be found, a different solution configuration is chosen and the process is repeated from the start of the seam. If none of the possible solution configurations leads to a feasible trajectory, then the workpiece must be repositioned to be successfully welded.

Inter-weld paths are defined as simple point-to-point, joint-interpolated motions. The starting and endpoints of adjacent welds define the endpoints of this path. Additional positions can be inserted into the path to avoid interference.

More advanced methods for path planning based on optimization of redundant systems are briefly discussed.

6. Graphics methods

A hidden surface removal algorithm has been developed to improve the graphical depiction of the workstation. Separating planes are used to sort disjoint convex polyhedra by depth. The method is too slow for real-time application, but it provides a useful means for clarifying confusing wire-frame scenes.

Speed and efficiency are important characteristics of interactive graphics software. The Fortran and C programming languages are compared for some simple benchmark tests, and Fortran is found to run faster on the IRIS computer. Speed can be increased by up to an order of magnitude by upgrading the hardware. Careful design of algorithms is essential for the achievement of maximum efficiency.

7. AUTOWELD software package

An interactive graphics software package, AUTOWELD, has been developed on a Silicon Graphics IRIS-1400 computer. AUTOWELD incorporates the results of this work, and demonstrates the feasibility of automatic generation of welding programs. AUTOWELD is highly interactive, and provides an informative graphical display. A complete welding program can be developed under the interactive guidance of the user.

8. Conclusions

The goals of this contract have been successfully met, and a powerful interactive graphics software package, AUTOWELD, has been developed. AUTOWELD is practical and useful, and clearly demonstrates the feasibility and value of further work in this area.

SUMMARY

This report presents results of work performed under contract for the National Research Council Canada, Industrial Materials Research Institute[1]. The central problem is the automatic determination of feasible robot trajectories to weld seams located on a given workpiece, based on a model of the system.

Several related problems are solved in developing these results:

1. A method is developed for modelling the geometry of the workstation, as well as the relevant seam and welding information, and the required kinematic specifications. The model is intended to be derived from an externally generated CAD data base.

2. An explicit inverse kinematics solution for a six-degree of freedom robot is used. The workpiece position must be fixed or known as a function of the distance along the weld. The set of solutions that correctly position the welding torch relative to the weld can be determined.

3. Four interference detection methods are developed and assessed and the best method is chosen. Several mathematical techniques and software algorithms have been developed to maximize the efficiency and speed of these methods.

---

[1] DSS File Number 16SD-31950-4-0026.

4. An iterative search algorithm finds the first feasible trajectory for the case of a six degree of freedom robot and a workpiece whose position is known for each point along the weld seam. If no feasible trajectory is found, the user must redefine the workpiece position and the process is repeated. More general methods of path planning for systems with redundant degrees of freedom are briefly discussed.

5. The robot trajectories between welds are limited only by the required torch positions at the ends of the welds. Additional robot positions can be inserted between these endpoints to define a collision-free joint-interpolated trajectory.

6. Graphical methods are developed for hidden surface removal and fast graphical display.

The results have been incorporated into a comprehensive software package called AUTOWELD, implemented in Fortran on a Silicon Graphics IRIS-1400 computer. AUTOWELD is an interactive graphics system that allows a user to develop a complete robot welding program at the terminal. The user has great control in defining and modifying the task, and can verify the results at any stage.

CONTENTS

EXECUTIVE SUMMARY i

SUMMARY v

1. INTRODUCTION 1

2. PROBLEM DEFINITION 2

3. DESCRIPTION OF THE WORKSTATION 4

    3.1 CAD Description of the Workpiece 4

        3.1.1 Geometric Modelling 4

        3.1.2 CAD Seam Description 8

    3.2 Description of the Robot and Positioning Table 9

        3.2.1 Link Geometry and Dimensions 12

4. KINEMATICS 16

    4.1 Review of Homogeneous Transformations 16

        4.1.1 Homogeneous Coordinate Transformation 16

        4.1.2 Serial Combination of Homogeneous Transformations 18

    4.2 Statement of Kinematics Problem 21

    4.3 Intermediate Homogeneous Coordinate Frames 21

        4.3.1 Table Frame Relative to World Frame 21

        4.3.2 Workpiece Frame Relative to Table Frame 23

        4.3.3 Weld Seam Frame Relative to Workpiece 24

        4.3.4 Tool Frame Relative to Weld Seam 24

    4.4 Calculation of the Tool Position and Orientation Relative to World ...27

5. PATH FEASIBILITY 28

    5.1 Statement of Feasibility Problem 28

5.2 Kinematic Feasibility 28

5.2.1 Definition of Robot Joint Frames and Resulting Transformations 30

5.2.2 Explicit Inverse Kinematics Solution for a PUMA Robot 31

5.2.3 Comparison of Five and Six Degree of Freedom Robots 42

5.3 Interference Avoidance .... .43

6. INTERFERENCE DETECTION 47

6.1 Face Splitting Method for Interference Detection 48

6.1.1 A Hierarchy of Tests 49

6.1.2 The Proposed Algorithm 50

6.1.3 Finding the Volume of Intersection 51

6.1.4 Implementation and Evaluation 52

6.1.5 Some Possible Improvements 54

6.2 Two Interference Methods Developed by J.P. Duncan 55

6.3 Separating Plane Method for Interference Detection 62

6.3.1 A Fast Search for a Separating Plane 62

6.3.2 Best Separating Face 63

6.3.3 Frame Coherence 63

6.3.4 Calculation of DMIN (P) 63

6.3.5 Steepest Descent Algorithm 67

6.3.6 Finding the Optimum Plane, and the Shortest Separating Distance 67

6.3.7 Calculation of Vertex to Face Distance 68

6.3.8 Edge to Edge Separating Distance 71

6.3.9 Projection of a Point onto a Line 76

6.3.10 Possible Improvements 78

6.4 Comparison and Assessment of Interference Detection Methods 78

7. PATH PLANNING 83

7.1 Iterative Search for a Feasible Welding Path 83

7.1.1 Extension to a Moving Workpiece 84

7.2 Inter-weld Path 85

7.3 Path Planning for Kinematic Systems with Redundancy 86

7.3.1 Redundancy and Local Optimization 88

7.3.2 Global Path Planning 89

8. GRAPHICS METHODS 91

8.1 Hidden Surface Removal Algorithm 91

8.2 Optimization for Speed 92

8.2.1 Fortran vs. C 93

8.2.2 Hardware 93

8.2.3 Efficiency of Algorithms 94

9. DESCRIPTION AND ASSESSMENT OF SOFTWARE PACKAGE 'AUTOWELD' 95

9.1 Design Approach 95

9.2 Features and Capabilities 95

9.3 Performance Assessment for Several Test Cases 99

9.4 Auxiliary Software 99

9.4.1 GEM - A Solid Modelling Program 99

9.4.2 GENPRO - A Post-Processor to Generate a Pseudo-program for Welding 99

10. CONCLUSIONS...104

10.1 Analysis and Discussion of Work Completed 104

10.1.1 Geometric modelling 104

10.1.2 CAD seam specification 104

10.1.3 Kinematics 104

10.1.4 Interference detection 105

10.1.5 Expert welder 105

10.1.6 Hidden surface removal 105

10.1.7 User Interaction 106

10.2 Suggestions for Further Work 106

11. REFERENCES 108

APPENDIX I - THE FACE SPLITTING INTERFERENCE DETECTION METHOD 111

I.1 DATA STRUCTURE 111

I.1.1 Static Data Structure 111

I.1.2 Dynamic Data Structures 114

I.1.3 Data Structure for Interference Detection 114

I.2 DESCRIPTION OF MATHEMATICAL METHODS 117

I.2.1 Surface Shading 117

I.2.2 Testing for Possible Face Intersection 118

I.2.3 Testing for Overlap in Screen Coordinates 118

I.2.4 Intersection of a Polygon by a Plane 120

I.2.5 Determination of Interior Faces 124

I.3 SOFTWARE STRUCTURE 125

I.4 LIST OF SUBROUTINES 126

APPENDIX II - DUNCAN INTERFERENCE DETECTION METHODS 130

APPENDIX III - SOFTWARE DOCUMENTATION OF 'AUTOWELD' 174

III.1 FUNCTIONAL DESCRIPTION OF 'AUTOWELD' 174

III.1.1 Subroutines Grouped by Function 175

III.2 'AUTOWELD' USERS GUIDE 177

III.2.1 Compiling AUTOWELD 177

III.2.2 Input Files 177

III.2.3 Detailed Contents of User-defined Files 178

III.2.4 Running AUTOWELD 192

III.3 SUBROUTINE DESCRIPTIONS 199

III.3.1 Subroutine Source Libraries 213

III.3.2 Subroutines Which Call the IRIS Graphics Library 215

III.4 DATA STRUCTURES 216

III.4.1 External Data Files 216

III.4.2 Internal Data Structures 221

III.4.3 Common Blocks 229

III.4.4 Dimensioning of Variables 236

III.5 SUGGESTED IMPROVEMENTS 242

APPENDIX IV - CAD FILE GENERATION MODULE PROGRAM GEM 244

IV.1 CADFILE GENERATION MODULE PROGRAM GEM - ANALYSIS AND PROGRAM DESIGN 244

IV.1.1 Reference Frames, Vectors, Variables and Equations 245

IV.1.2 Equations 246

IV.1.3 Geometric Definition of Parts 248

IV.1.4 Program Design 248

IV.1.5 Symbol Table for Program GEM and Subroutine PART 248

IV.2 CADFILE GENERATION MODULE PROGRAM GEM - USERS GUIDE 258

APPENDIX V - SOFTWARE SOURCE LISTINGS 261

V.1 AUTOWELD SOURCE LISTINGS 261

V.2 GEM SOURCE LISTINGS 261

V.3 GENPRO SOURCE LISTING 261

V.4 FACE SPLITTING SOURCE LISTINGS 261

V.5 DUNCAN METHODS SOURCE LISTINGS 261

V.6 FORTRAN AND C BENCHMARK PROGRAMS 261

LIST OF FIGURES

Figure 3-1 Hierarchical Representation of Topology 6

Figure 3-2 Minimal Geometry Data Structure 7

Figure 3-3 Representation of the CAD Seams as a Network of Segments 10

Figure 3-4 User Seam Definitions 11

Figure 3-5 Kinematics of a General Workstation 13

Figure 3-6 Link Geometry and Frames for a Revolute Joint 14

Figure 3-7 Link Geometry and Frames for a Prismatic Joint 15

Figure 4-1 Coordinate Transformation 17

Figure 4-2 Serial Combination of Homogeneous Transformations 19

Figure 4-3 Homogeneous Coordinate Frame Defined by n, o, a, p 20

Figure 4-4 Series of Homogeneous Coordinate Frames 22

Figure 4-5 Weld Frame Relative to Workpiece 25

Figure 4-6 Tool Frame Relative to Weld Seam 26

Figure 5-1 PUMA Joint Frames and Link Parameters 34

Figure 5-2 Relative Geometry of PUMA Link Frames F_3, F_4, F_5, F_6 ..37

Figure 5-3 Rotation of Tool Frame About Tool Axis 44

Figure 6-1 Photographs of Screen Output 53

Figure 6-2 Duncan Interference Method I 58

Figure 6-3 Duncan Interference Method II 59

Figure 6-4 Example of a Polyhedron Not Represented by a Mercator's Projection 60

Figure 6-5 A Connected Graph Representation 61

Figure 6-6 Faces Defining Separating Planes 64

Figure 6-7 No Face Defines a Separating Plane 65

Figure 6-8 A Steepest Descent Search for the Closest Vertex 66

Figure 6-9 The Shortest Separating Vector Between Two Polyhedra 69

Figure 6-10 Separation and Interference Cases 70

Figure 6-11 A Face Inclusion Test 72

Figure 6-12 Face Edges and Vertex Edges 74

Figure 6-13 The Common Normal of Two Lines 75

Figure 6-14 Shortest Connecting Vectors for Two Edges 77

Figure 7-1 Generation of an Inter-Weld Path 87

Figure 9-1 Sketches of the test Workpieces 100

Figure I-1 Geometric Data Structure 112

Figure I-2 Dynamic Data Structure 115

Figure I-3 Definition of a Plane 119

Figure I-4 Separating Line between Disjoint Polygons 121

Figure I-5 Best Separating Line 122

Figure I-6 Determination of Interior Faces 123

Figure IV-1 Reference Frames and Vectors 249

Figure IV-2 Box Faces and Normals 250

Figure IV-3 Cylinder Faces and Normals 251

Figure IV-4 Flowchart for Program GEM 252

Figure IV-5 Flowchart for Program GEM (continued) 253

Figure IV-6 Flowchart for Subroutine PART 254

Figure IV-7 Flowchart for Subroutine PART (continued) 255

1. INTRODUCTION

The work described in this report is the second phase of a major project in welding robot automatic programming initiated by the Canadian National Research Council Industrial Materials Research Institute (IMRI). The Phase 1 report [Buchal et al 1984] described the development of solutions and programs for the inverse kinematics, welding reference and path feasibility components of an automatic programmer package. These components were demonstrated on a restricted capability graphics terminal. At a review meeting[1] held at UBC on January 24, 25, 1984, the necessary research tasks for the next phase of the work were identified. A formal proposal was subsequently prepared[2]. The major goals identified in the proposal are to develop, evaluate and select an optimum technique for detection of interference between objects in the workstation, to develop a path planning technique recognizing inverse kinematics and interference constraints and to demonstrate the complete package on an advanced graphics computer terminal. The following report describes how these goals have been successfully met. All aspects of the software development are covered, including relevant work from the Phase 1 report.

---

1. Review meeting held at the University of British Columbia; Department of Mechanical Engineering, January 24, 25, 1984; Attendees: Dr. A. Hughes Marquis (IMRI), Mr. Jean-Paul Bullod (IMRI), Dr. D. Cherchas (UBCME), Dr. F. Sassani (UBCME), Mr. R. Buchal (UBCME), Mr. W. Cameron (UBC TRUIMF), Mr. H. Eppich (EBCO Industries, Richmond).

2. "Proposal for Development of Techniques and Software for Interference Detection and Path Planning for a Welding Robot Automatic Programming System", University of British Columbia Department of Mechanical Engineering proposal to IMRI, November 21, 1984.

2. PROBLEM DEFINITION

Current industrial practice requires that robots be programmed manually for each new task. The programming is carried out using a teach pendant, or by physically guiding the robot through its desired motions. The difficulty of generating new programs restricts robot use to simple, repetitive tasks.

The aim of this project is to develop methods for generating robot welding programs off-line, and to integrate them into a useable software package. In particular, the following problems have been identified:

a. Develop a geometric modelling method for representing objects in the workstation.

b. Specify a format for an externally generated CAD description of the seams.

c. Given the workpiece position and digitized information about the seam and workpiece, solve the inverse kinematics problem to generate a robot trajectory.

d. Develop and assess several methods for detecting interference between the robot and the workpiece or positioning table.

e. Develop a strategy for finding a feasible trajectory which avoids interference and does not exceed physical joint limits.

f. Allow uniform workpiece motion to be specified during a weld.

g. Develop a method for generating robot trajectories between welds.

h. Develop graphical tools and techniques to allow display of the robot, workpiece and positioning table, with interactive user control.

i. Allow the operator to specify the task completely, and to modify and verify the resulting program interactively.

j. Incorporate a previously developed expert welder system to automatically generate the welding parameter values. The further development of an expert welder is not an aim of this work.

k. Optimize the speed and response of the software package.

Solutions to some of these problems have been developed under a previous contract [Buchal et al 1984], [Buchal 1984]. The inverse kinematics solutions and feasible path search strategy have been adapted without major changes. Other previous work has been heavily modified.

The desired product of this work is a software package, AUTOWELD, incorporating the solutions and methods developed. The program will take a CAD description of the workpiece and workstation and with user interaction generate a complete robot and positioning table program to successfully execute the specified welding task.

AUTOWELD is to be developed to the stage where the successful solutions to the above problems can be clearly demonstrated. AUTOWELD must be general and flexible to allow further development without major revisions.

3. DESCRIPTION OF THE WORKSTATION

3.1 CAD Description of the Workpiece

3.1.1 Geometric Modelling. For graphical depiction and path planning, a world model must be formulated which contains complete geometric descriptions of all objects in the workstation. There are two common methods for representing objects:

1. Boundary Representation.

   With this method, a solid is represented by a description of its surface. The most common form is the polyhedral model, which represents a polyhedral approximation of a solid by vertices, edges and faces [Hosaka, Kimura and Kakishita 1974], [Wesley et al 1980]. Redundant information can be included to speed processing and eliminate ambiguity. The resulting model is simple and general.

2. Constructive Solid Geometry

   More advanced CAD systems generate complex solids by performing boolean operations and transformations on a set of simple solid primitives such as boxes and cylinders [Braid 1975], [Requicha 1980]. The resulting representation contains a record of the composite primitives and the sequence of operations.

More complete surveys of solid representations can be found in the literature [Baer, Eastman and Henrion 1979], [Requicha 1980].

For simplicity and generality, we have chosen a polyhedral boundary representation for the objects in the workstation. The solid objects are modelled by clusters of convex polyhedral solids. These models are approximate only, but the error can be reduced to any desired level by increasing the number of polyhedral faces. Conversely, very coarse approximations can be used when appropriate. In general, a trade-off is made between processing speed and representational accuracy. For engineering shapes, an accurate representation is often possible with a simple polyhedral model.

The boundary representation has two parts: a description of the topology, and a specification of metrics. The topology defines generic shapes such as boxes and pyramids. The representation is completed by specifying the actual coordinates of the shape boundaries. Redundant information is included in the topological representation in order to maximize processing speed. In other words, information which could be derived from a minimal description is stored permanently for fast access when needed.

The topological representation is hierarchical in structure as shown in Fig. 3.1. The minimal information is shown below:

- list of clusters in the world
- list of parts in each cluster
- list of faces on each part
- list of vertices on each face.

The corresponding metric information consists of:

- a table of vertex coordinates.

A minimal geometry data structure is shown in Fig. 3.2.

The following redundant information, which can be derived from the minimal representation, is also stored:

- faces adjacent to faces
- faces adjacent to vertices

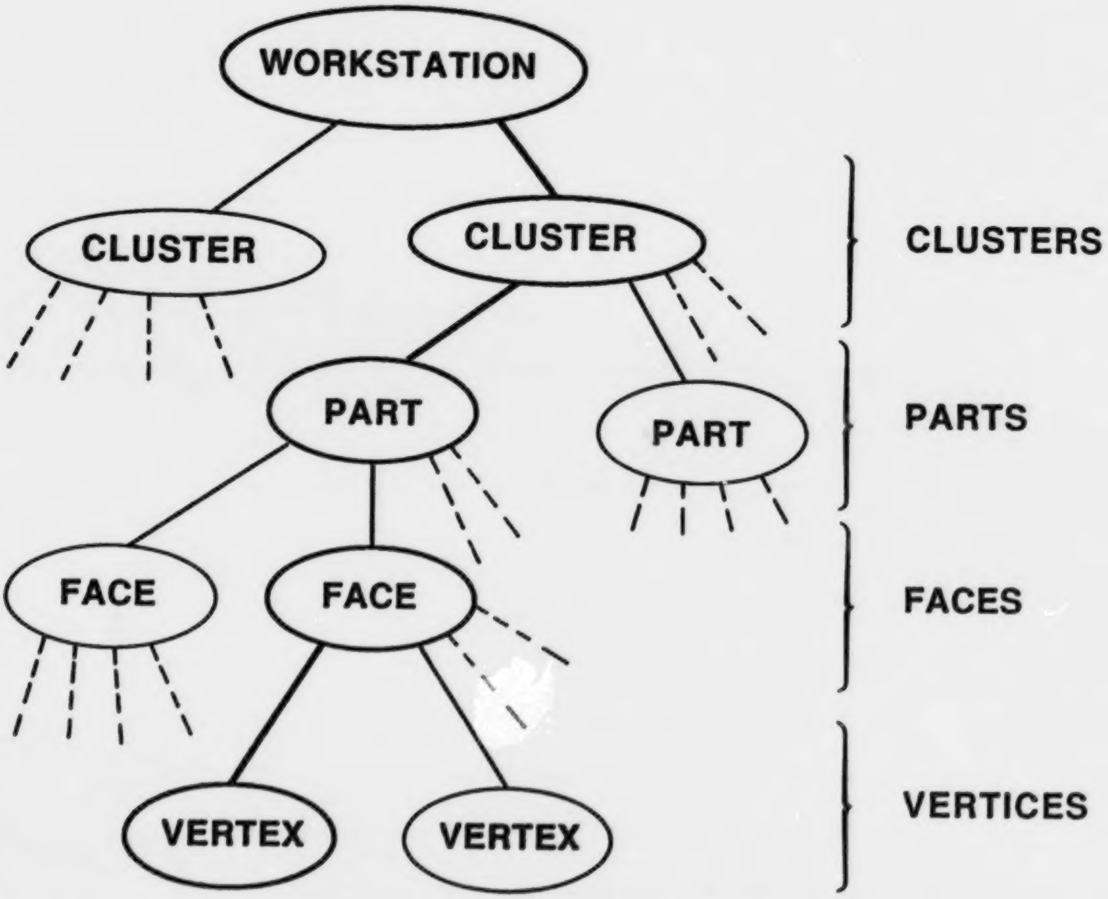
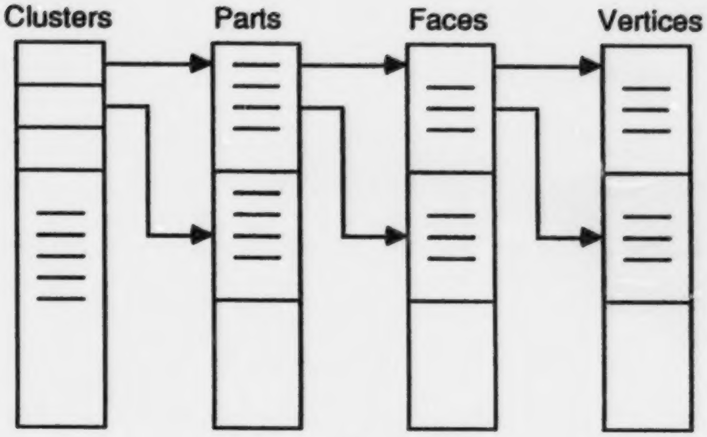
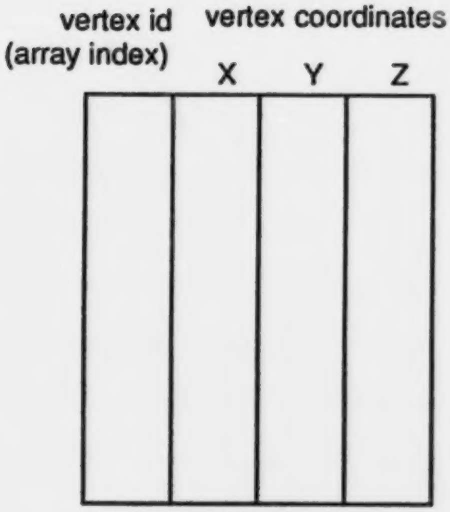

WORKSTATION
CLUSTER
CLUSTER
PART
PART
FACE
FACE
VERTEX
VERTEX
CLUSTERS
PARTS
FACES
VERTICES

Figure 3-1. Hierarchical Representation of Topology


Clusters
Parts
Faces
Vertices

(a) TOPOLOGY


vertex id
(array index)
vertex coordinates
X
Y
Z

(b) METRIC DATA

Figure 3-2. Minimal Geometry Data Structure.

- vertices adjacent to vertices
- list of faces on a cluster
- list of vertices on a cluster
- table of face normals.

Each polyhedron is defined relative to a coordinate frame fixed to its parent cluster. The polyhedral representations of solids are positioned and moved in space by transforming the metric part of the representation. In other words, only the vertex coordinates and face normals are transformed. The untransformed reference coordinates are stored at all times, and the transformed coordinates are generated by transforming the reference coordinates by the appropriate net transformation or product of transformations.

3.1.1.1 Separating Planes For hidden surface removal and interference detection, separating planes between pairs of parts must be found. In many cases, such as when a number of parts are defined relative to a single cluster frame and do not move relative to each other, fixed separating planes can be specified. These separating planes are transformed along with the clusters, and are available without further calculation. Separating planes are discussed in more detail in Section 5.3 and in Appendix III.4.2.3.

3.1.2 CAD Seam Description The CAD seam description is part of the externally generated CAD specification of the workpiece. The workpiece has one or more seam 'segments', which are defined as space curves. These segments are connected by nodes to form a topological network as shown in

Fig. 3.3. The seam segments are represented by a series of closely spaced data points containing the coordinates of the curve relative to the workpiece reference frame. At each point on the seam, the unit surface normals of the two adjacent parent surfaces are also known. This CAD data is processed to generate the seam tangent, normal, and lateral vectors defining a coordinate frame at each point.

Based on this minimal representation, the user can define starting and ending points of welds, the order of the welding, and the welding direction. To build up a definition of a weld seam, the user interactively picks points on the CAD reference seam. The user must pick the starting point, any nodes where the seam branches, and the end point. The resultant seam specification is a composite of CAD seam segments as shown in Fig. 3.4.

The CAD seam description also contains welding specifications for each segment. The seam type, weld cross sectional area, and thicknesses of the parent material are specified. This information is used by the expert welder to generate output welding parameters.

3.2 Description of the Robot and Positioning Table

For a system with a continuously movable positioning table, the robot and table can be described as general kinematic chains. The robot and table kinematics are fully specified by the following parameters:

1. Number of degrees of freedom.
2. Complete link geometry and dimensions.
3. Specification of joint types (revolute or prismatic).
4. Physical joint limits.

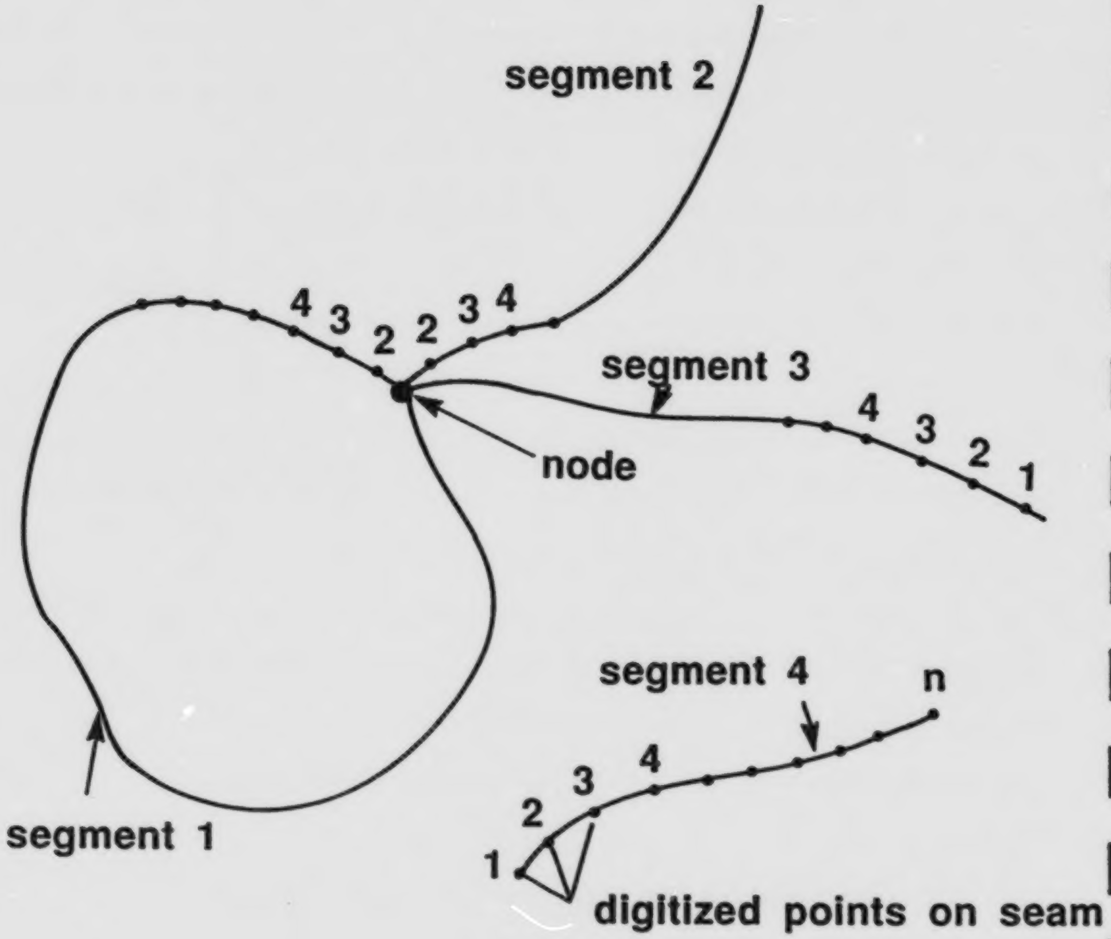

segment 2
4
3
2
2
3
4
segment 3
4
3
2
1
node
segment 4
n
4
3
2
1
segment 1
digitized points on seam

Figure 3-3. Representation of the CAD Seams as a Network of Segments.

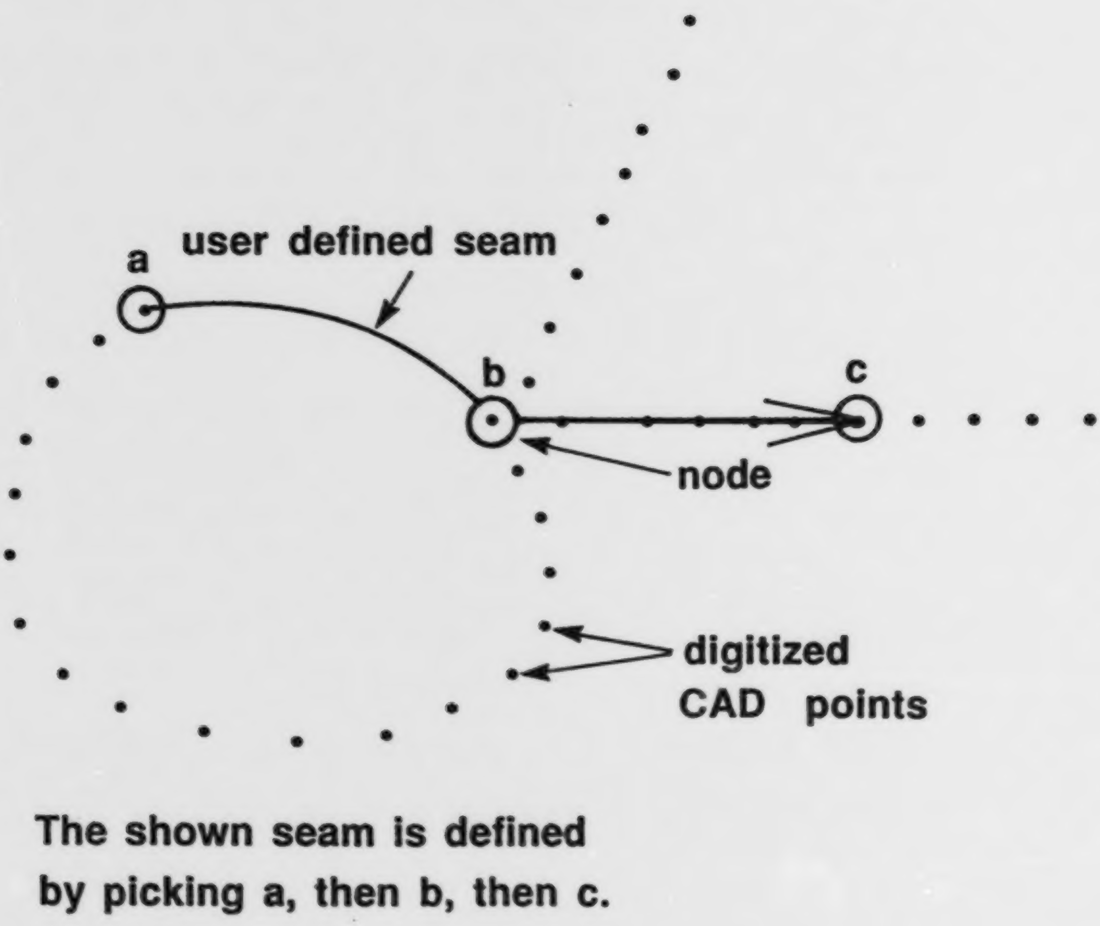

a
user defined seam
b
c
node
digitized
CAD points
The shown seam is defined
by picking a, then b, then c.

Figure 3-4. User Seam Definitions

5. Fixed transformation from torch or table frame to last link frame.
6. Fixed transformation from robot or table base to world frame.
7. Default configuration when more than one inverse solution exists.

3.2.1 Link Geometry and Dimensions

We will follow the conventions of Denavit and Hartenberg [1955] in describing link geometry and dimensions. We can define an n degree of freedom robot as a series of n rigid links connected by n joints (Figs. 3.6, 3.7). The i^{th} link is characterized by a length a_i and a twist angle α_i. a_i is the length of the common normal $\underline{a}_i$ between the two joint axes, and α_i is the angle between the axes in a plane perpendicular to $\underline{a}_i$.

The relative displacement between $\underline{a}_{i-1}$ and $\underline{a}_i$ at the joint i axis is d_i, the distance between the links. θ_i is the angular displacement of $\underline{a}_i$ and $\underline{a}_{i-1}$ about the joint i axis and is called the angle between the links.

Parameters a_i and α_i are fixed for a given link. For a prismatic joint where motion is along the joint axis, d_i varies and θ_i is fixed. For a revolute joint, d_i is fixed and θ_i varies.

Given these parameters, we can completely define the kinematics of the robot. The parameters are stored in a file.

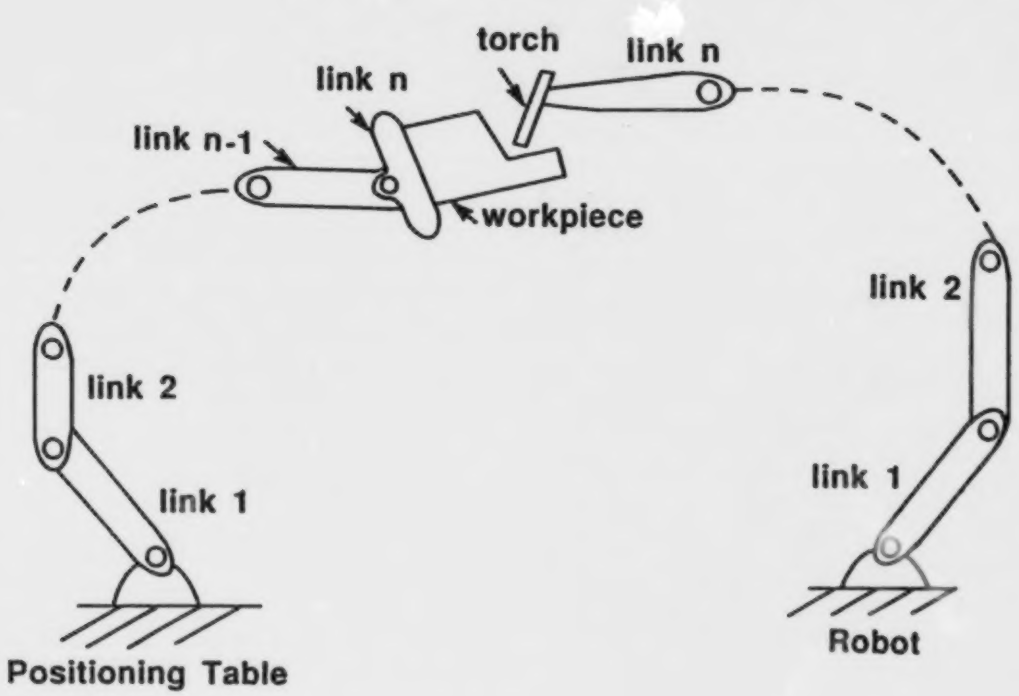

torch
link n
link n
link n-1
workpiece
link 2
link 2
link 1
link 1
Positioning Table
Robot

Figure 3-5. Kinematics of a General Workstation

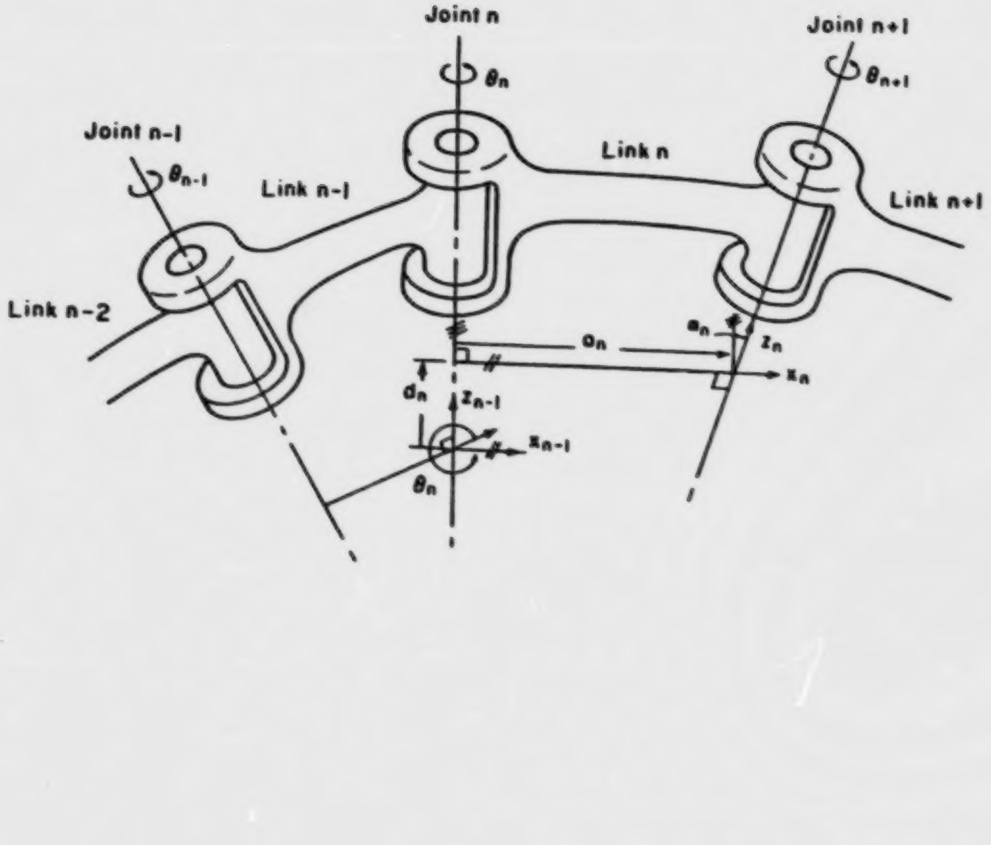

Joint n
θ_n
Joint n+1
θ_{n+1}
Joint n-1
θ_{n-1}
Link n
Link n-1
Link n+1
Link n-2
a_n
α_n
z_n
x_n
d_n
z_{n-1}
x_{n-1}
θ_n

Figure 3-6. Link Geometry and Frames for a Revolute Joint.

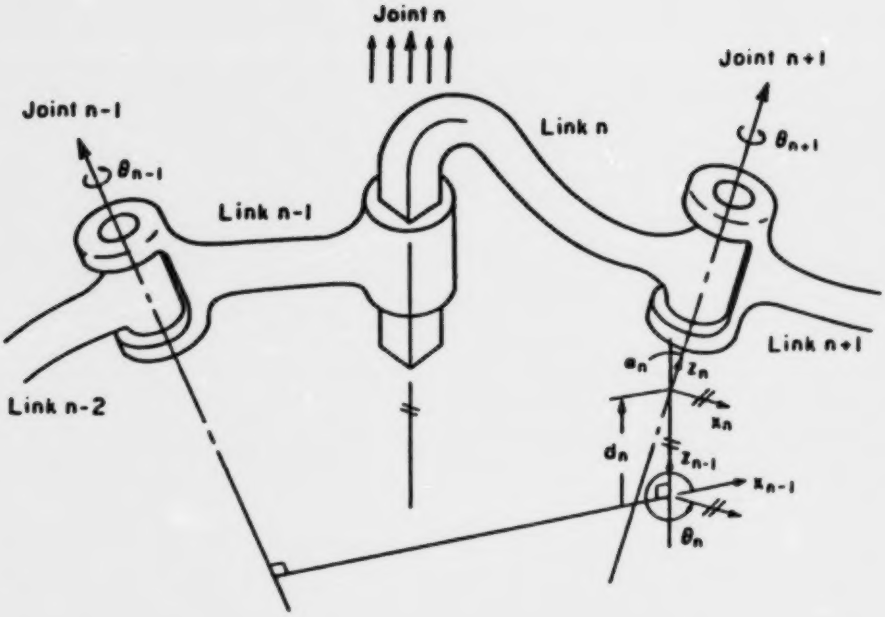

Joint n
Joint n+1
Joint n-1
θ_{n+1}
θ_{n-1}
Link n
Link n-1
Link n+1
Link n-2
a_n
z_n
x_n
d_n
z_{n-1}
x_{n-1}
θ_n

Figure 3-7. Link Geometry and Frame for a Prismatic Joint.

4. KINEMATICS

4.1 Review of Homogeneous Transformations

4.1.1 Homogeneous Coordinate Transformation

It is often useful to transform the coordinates of a point in space from one coordinate frame to another (Fig. 4.1). A rotational transformation is required if corresponding axes of two coordinate frames F_{i-1}, F_i are not parallel. A translational transformation is required if the frame origins are not coincident. In general, both a rotation and a translation are required.

A homogeneous coordinate transformation allows us to perform both rotation and translation in a single operation using a 4×4 homogeneous transformation matrix. For example, a vector $\underline{r}$ expressed relative to and projected onto a coordinate frame F_i can be transformed into a vector $\underline{R}$ relative to and projected onto a frame F_{i-1} by the following matrix multiplication:

$$\underline{R} = \underline{H}_{i-1,i}\,\underline{r} \ , \tag{4.1}$$

where $\underline{H}_{i-1,i}$ is a 4×4 homogeneous transformation matrix relating frame F_i to Frame F_{i-1}. The vectors $\underline{R}$ and $\underline{r}$ are homogeneous 4×1 column vectors, where the first three elements are the cartesian coordinates and the last element is a scaling factor. For our purposes this scaling factor is always set to 1.

$\underline{H}_{i-1,i}$ has the following form:

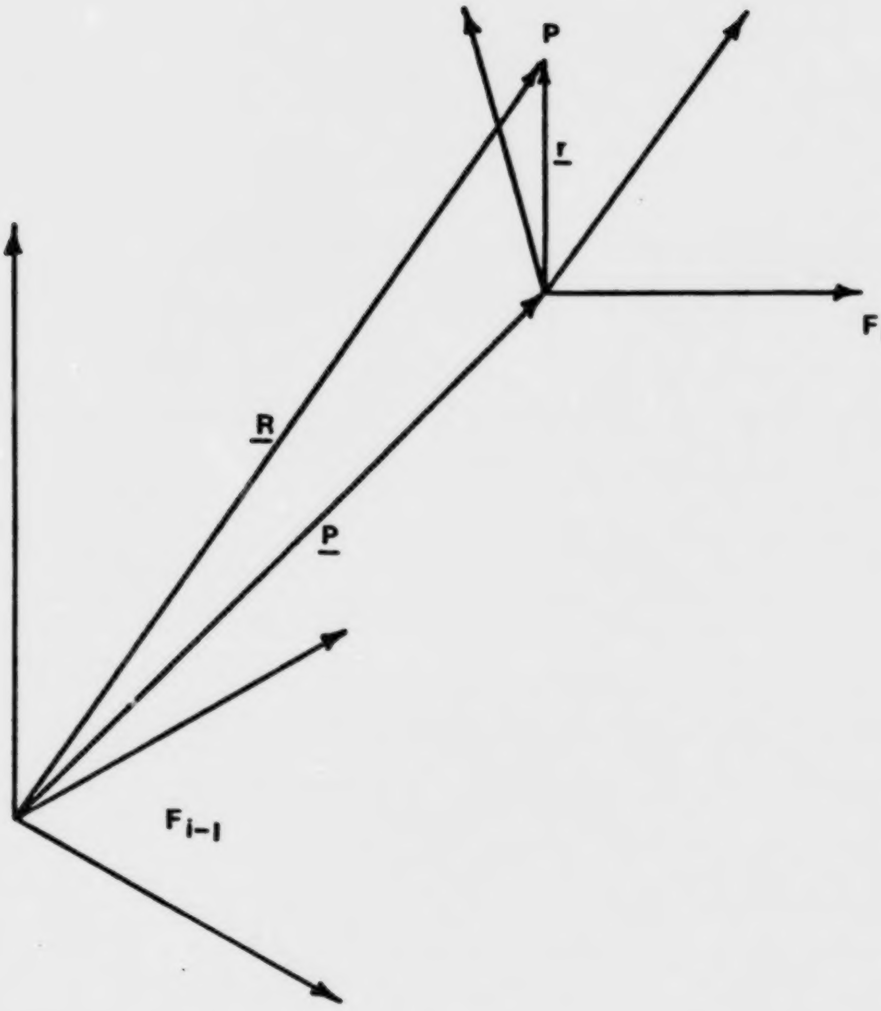


Figure 4-1. Coordinate Transformation.

$$\underline{H}_{i-1,i} = \begin{array}{c} \begin{array}{cc} \text{rotation} \qquad\qquad & \text{translation} \end{array} \\ \left[\begin{array}{ccc|c} \multicolumn{3}{c|}{3 \times 3} & P_x \\ \multicolumn{3}{c|}{\text{rotation matrix}} & P_y \\ & & & P_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \end{array} . \qquad (4.2)$$

A more detailed description of homogeneous transformations can be found in other references [Paul 1981].

4.1.2 Serial Combination of Homogeneous Transformations

We can combine homogeneous transforms in series as shown in Fig. 4.2.

Suppose we want to find $\underline{R}$ given $\underline{r}$. If we know $\underline{H}_{1,2}$ and $\underline{H}_{2,3}$ we can find $\underline{H}_{1,3}$ as the product of $\underline{H}_{1,2}$ and $\underline{H}_{2,3}$:

$$\underline{H}_{1,3} = \underline{H}_{1,2}\, \underline{H}_{2,3} \ . \qquad (4.3)$$

Thus,

$$\underline{R} = \underline{H}_{1,3}\, \underline{r} \ . \qquad (4.4)$$

4.1.3 Serial Combination of Homogeneous Frames

We can show that the homogeneous transformation $H_{i-1,i}$ defines a coordinate frame F_i relative to F_{i-1} .

$\underline{H}_{i-1,i}$ can be written as

$$\underline{H}_{i-1,i} = [\ \underline{n},\ \underline{o},\ \underline{a},\ \underline{P}\ ] \qquad (4.5)$$

where $\underline{n}$, $\underline{o}$, $\underline{a}$, $\underline{p}$ are 4 × 1 column vectors. $\underline{n}$, $\underline{o}$, $\underline{a}$ define the basis vectors of F_i relative to F_{i-1}, and $\underline{P}$ is the origin of F_i relative to F_{i-1}, as shown in Fig. 4.3.

Note that if we know the position and orientation of Frame F_i relative to F_{i-1}, $\underline{H}_{i-1,i}$ is also known.

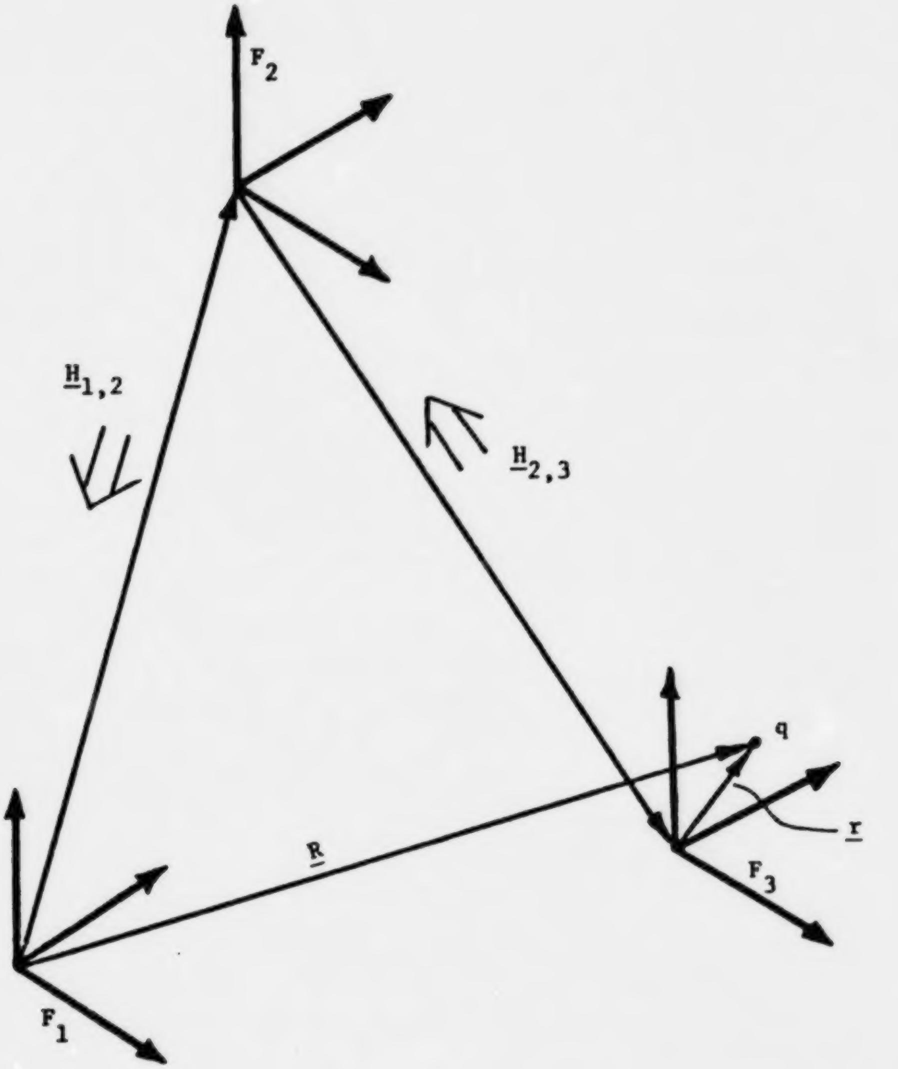

F_2
$\underline{H}_{1,2}$
$\underline{H}_{2,3}$
q
$\underline{r}$
$\underline{R}$
F_3
F_1

Figure 4-2. Serial Combination of Homogeneous Transformations.

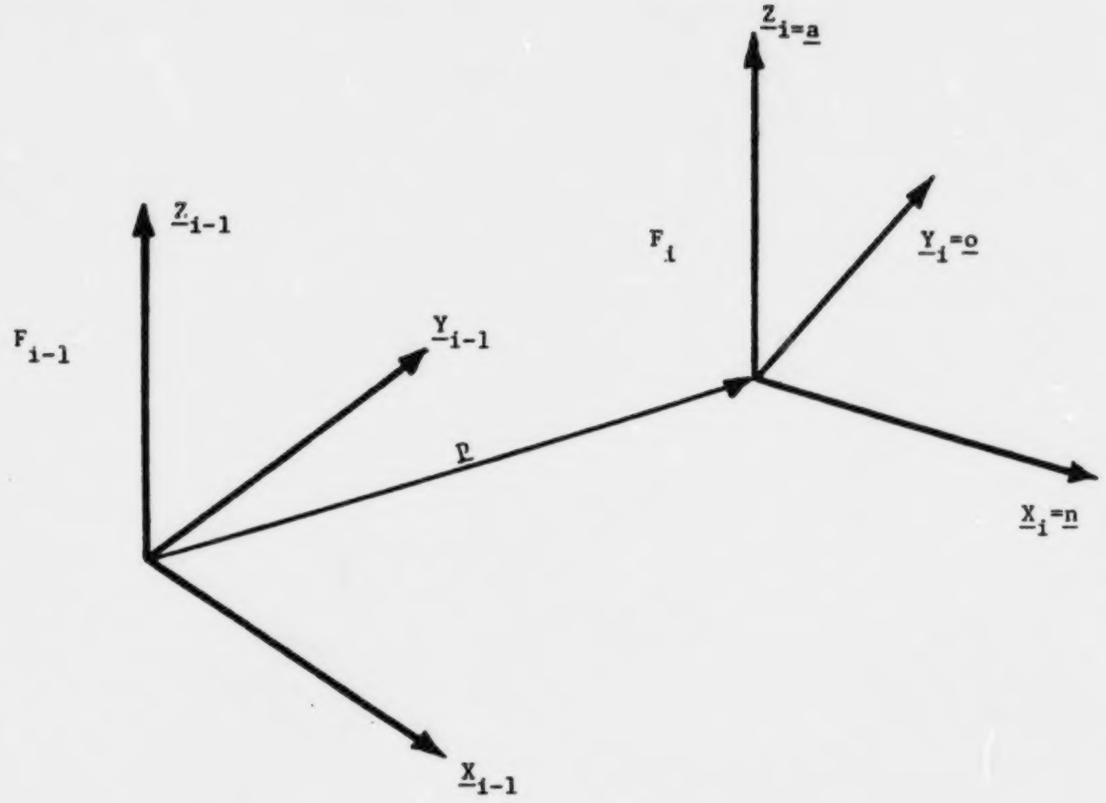

$\underline{Z}_{i=\underline{a}}$
$\underline{Z}_{i-1}$
F_i
$\underline{Y}_i=\underline{o}$
$\underline{Y}_{i-1}$
F_{i-1}
$\underline{p}$
$\underline{X}_i=\underline{n}$
$\underline{X}_{i-1}$

Figure 4-3. Homogeneous Coordinate Frame Defined by $\underline{n}$, $\underline{o}$, $\underline{a}$, $\underline{p}$.

4.2 Statement of Kinematics Problem

To generate a robot program, we must find the tool position and orientation at each point along the weld seam trajectory. Mathematically stated, we wish to find the homogeneous coordinate frame F_G of the tool relative to and projected onto the robot base coordinate frame F_R. F_G is defined by the homogeneous coordinate transformation $\underline{H}_{R,G}$ which takes us from F_G to F_R. Thus, we wish to find $\underline{H}_{R,G}$. We can define a string of intermediate coordinate frames F_i for which $\underline{H}_{i-1,i}$ can be calculated, as shown in Fig. 4.4.

We define the following homogeneous coordinate frames:

a) F_R - robot base frame.

b) F_T - positioning table frame.

c) F_B - workpiece frame.

d) F_C - weld seam frame.

e) F_G - tool frame.

We can solve for the homogeneous transformations $\underline{H}_{R,T}$, $\underline{H}_{T,B}$, $\underline{H}_{B,C}$, $\underline{H}_{C,G}$. We can then find $\underline{H}_{R,G}$.

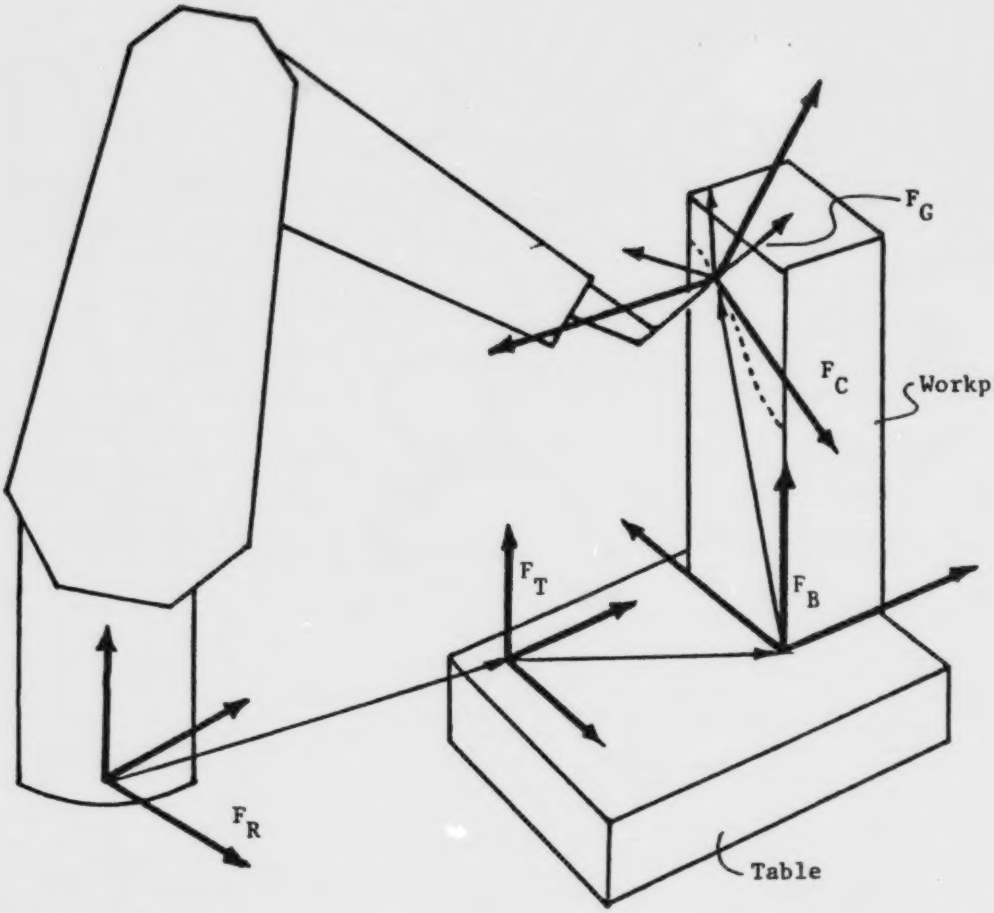
$$\underline{H}_{R,G} = \underline{H}_{R,T}\,\underline{H}_{T,B}\,\underline{H}_{B,C}\,\underline{H}_{C,G} \quad . \qquad (4.6)$$

4.3 Intermediate Homogeneous Coordinate Frames

4.3.1 Table Frame Relative to Robot Base

We wish to define the position of the table frame F_T relative to the robot base coordinate frame F_R. The problem is to find the transformation $\underline{H}_{R,T}$.

Suppose we analyze the table as an n-link system, similar to a robot.


F_G
F_C
Workpiece
F_T
F_B
F_R
Table

F_R	Robot Frame
F_T	Welding Table Frame
F_B	Workpiece Frame
F_C	Weld Seam Frame
F_G	Tool Frame

Figure 4.4 Series of Homogeneous Coordinate Frames.

We can then define the transformation $\underline{H}_{i-1,i}$ from link i-1 to link i by the Denavit-Hartenberg [1955] equations

$$\underline{H}_{i-1,i} = \begin{bmatrix} \mathrm{Cos}\theta_i & -\mathrm{Sin}\theta_i \mathrm{Cos}\alpha_i & \mathrm{Sin}\theta_i \mathrm{Sin}\alpha_i & a_i \mathrm{Cos}\theta_i \\ \mathrm{Sin}\theta_i & \mathrm{Cos}\theta_i \mathrm{Cos}\alpha_i & -\mathrm{Cos}\theta_i \mathrm{Sin}\alpha_i & a_i \mathrm{Sin}\theta_i \\ 0 & \mathrm{Sin}\alpha_i & \mathrm{Cos}\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

where θ_i is the i^{th} joint angle and α_i and a_i are fixed link geometry parameters. Either θ_i (revolute joint) or d_i (prismatic joint) is variable.

We can combine the transformation for all n links as

$$\underline{H}_{0,n} = \underline{H}_{0,1} \underline{H}_{1,2} \cdots \underline{H}_{n-1,n} \quad . \tag{4.8}$$

This gives us the n^{th} link coordinate frame F_n relative to the first link coordinate frame F_0.

We also wish to specify the location of the table frame F_T relative to F_n, and the first link frame F_0 relative to the robot base frame F_R. These two transformations $H_{R,0}$, $H_{n,T}$ are fixed and known. Thus,

$$\underline{H}_{R,T} = \underline{H}_{R,0} \underline{H}_{0,n} \underline{H}_{n,T} \quad . \tag{4.9}$$

4.3.2 Workpiece Frame Relative to Table

We can safely assume that the workpiece is firmly attached to the table. Thus, $\underline{H}_{T,B}$ is some fixed rotation and translation relative to F_T which needs to be specified only once during initialization.

4.3.3 Weld Seam Frame Relative to Workpiece

We define the weld seam frame F_C as shown in Figure 4.5, where $\underline{T}$ is tangent to the seam trajectory and $\underline{A}$ bisects the surface normals $\underline{N}_1$, $\underline{N}_2$. (i.e., $\underline{A}$ is normal to weld face). The transformation $\underline{H}_{B,C}$ is then

$$\underline{H}_{B,C} = [\ \underline{T},\ \underline{B},\ \underline{A},\ \underline{R}\ ] \quad . \tag{4.10}$$

4.3.4 Tool Frame Relative to Weld Seam

The welding torch is positioned relative to the weld as required by standard welding practice. The position and orientation of the torch relative to the weld are characterized by angular offsets from the weld surface normal, α and β, and a tool gap ρ (Fig. 4.6). We know that the elements of $\underline{H}_{C,G}$ can be found from

$$\underline{H}_{C,G} = [\ \underline{n},\ \underline{o},\ \underline{a},\ \underline{P}\ ] \quad . \tag{4.11}$$

From trigonometry:

$$\underline{a} = \begin{bmatrix} -\sin\alpha \\ \sin\beta \\ -\cos\alpha\cos\beta \\ 0 \end{bmatrix} , \tag{4.12}$$

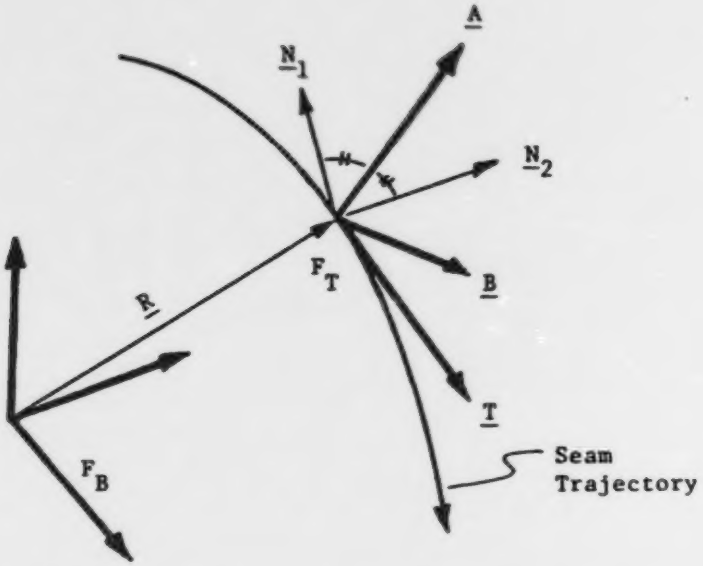
and $$\underline{P} = -\rho\,\underline{a} \quad . \tag{4.13}$$

$\underline{n}$, $\underline{o}$ are arbitrary -- let us define them as

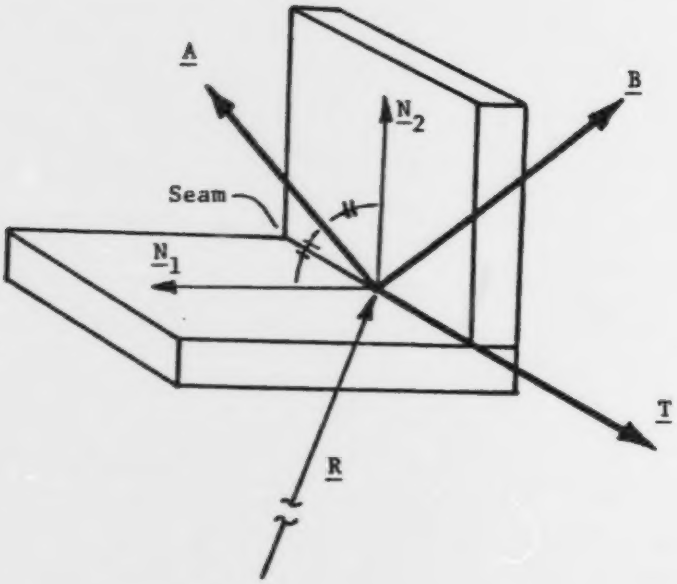
$$\underline{n} = \begin{bmatrix} 0 \\ \cos\beta\cos\alpha \\ \sin\beta \\ 0 \end{bmatrix} \tag{4.14}$$

and $$\underline{o} = \underline{a} \times \underline{n} \quad . \tag{4.15}$$

We can now find elements of $\underline{H}_{C,G} = [\underline{n}, \underline{o}, \underline{a}, \underline{P}]$.


A
N1
N2
B
T
R
FT
FB
Seam Trajectory

General Case


A
B
N2
Seam
N1
T
R

Example

Figure 4-5. Weld Frame Relative to Workpiece.

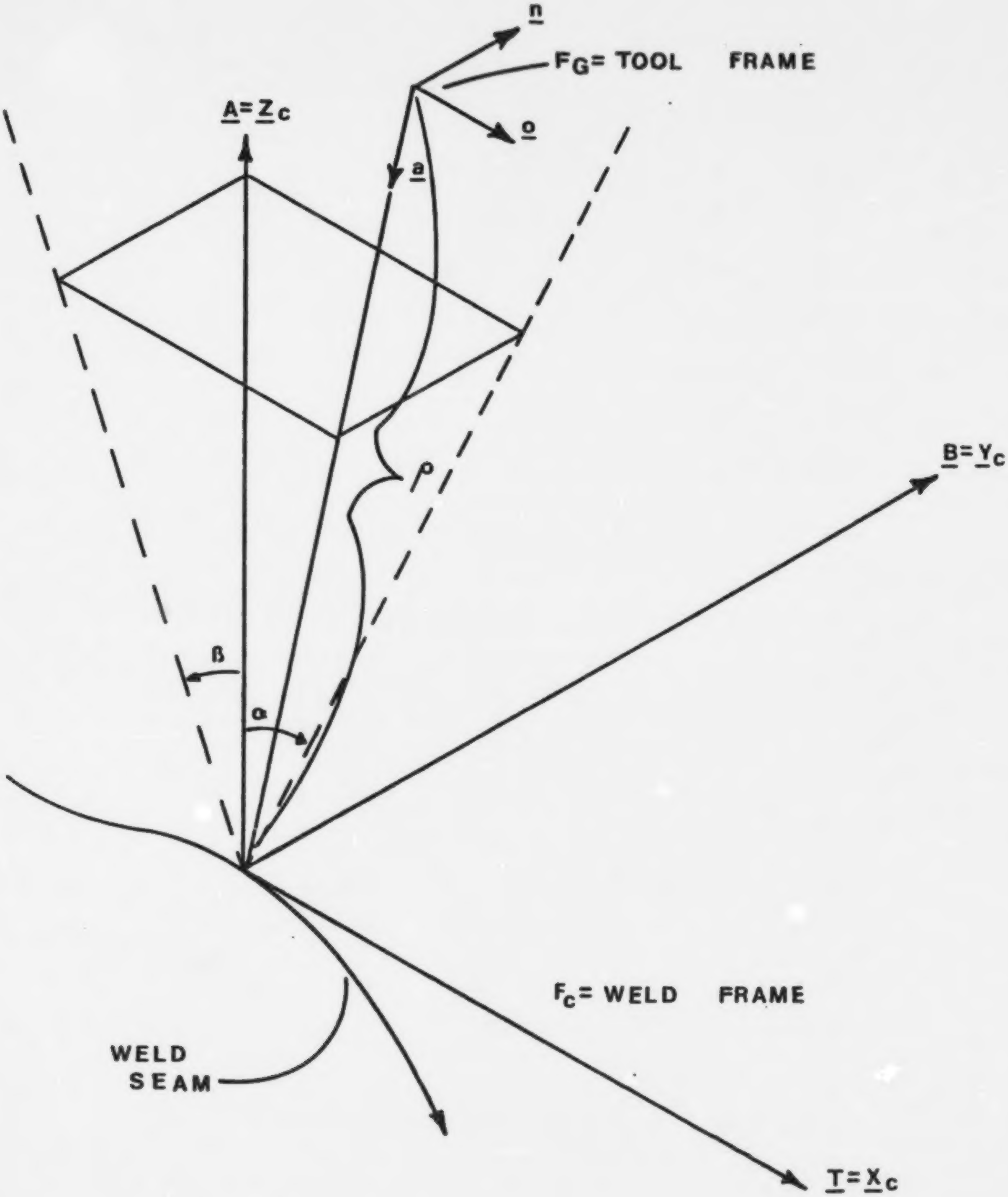

n
F_G = TOOL FRAME
A = Z_c
o
a
ρ
B = Y_c
β
α
F_c = WELD FRAME
WELD SEAM
T = X_c

Figure 4-6. Tool Frame Relative to Weld Seam.

4.4 Calculation of the Tool Position and Orientation Relative to World Coordinates

Once we have specified all the intermediate homogeneous transformations, we can calculate the tool position and orientation relative to F_R. This information is given by the homogeneous transformation $\underline{H}_{R,G}$ which takes us from the tool coordinates to robot coordinates. By combination of homogeneous transforms, we can calculate $\underline{H}_{R,G}$:

$$\underline{H}_{R,G} = \underline{H}_{R,T} \; \underline{H}_{T,B} \; \underline{H}_{B,C} \; \underline{H}_{C,G} \quad . \tag{4.16}$$

The elements of $\underline{H}_{R,G}$ have the form of four column vectors, $\underline{n}$, $\underline{o}$, $\underline{a}$, $\underline{P}$:

$$\underline{H}_{R,G} = [\, \underline{n}, \underline{o}, \underline{a}, \underline{P} \,] \quad . \tag{4.17}$$

We know that the vectors $\underline{n}$, $\underline{o}$, and $\underline{a}$ define the basis vectors of the tool coordinate frame F_G projected onto the robot base frame F_R, and $\underline{P}$ is the displacement of F_G relative to F_R. If we wish to define a world coordinate frame which is different from the robot base frame, they can be simply related by a fixed transformation.

5. PATH FEASIBILITY

5.1 Statement of Feasibility Problem

When we calculate a nominal robot tool trajectory through space, we may find that the trajectory is not feasible. The feasibility issue can be broken into two fundamental problems:

1. Kinematic constraints of the robot.
2. Interference or collision with objects in the workstation.

We can define a trajectory as being feasible if the robot can follow the trajectory smoothly, without violating the kinematic constraints and without colliding with or contacting any of the objects in the workstation.

Of the two problems we have defined, the kinematics problem is the least difficult, and we will discuss it first.

5.2 Kinematic Feasibility

Mathematically the conditions which must be satisfied for kinematic feasibility are:

1. The work envelope must not be exceeded.
2. The robot joint angles must remain within their physical limits.

In order for us to test these conditions it is necessary to solve for the joint angles as a function of the desired tool location and orientation relative to the robot base (i.e., $\underline{H}_{R,G}$). This is commonly known as the inverse kinematics solution. There are two particular cases which are of practical interest. Suppose we define a robot gripper state vector $\underline{X}$ defining n independent state variables. Let us also define a joint variable vector $\underline{\theta}$ defining m joint variables. If $n = m$, we can find a finite set of solutions of $\underline{\theta}$ for any set $\underline{X}$ in the work envelope. If $n < m$, an infinite set of solutions exists and we say the robot has redundant degrees of freedom.

The kinematic problem has been considered by many researchers over the years. Denavit and Hartenberg [1955] first formally specified the geometry of connected links with homogeneous matrices. This work has been fundamental to most later work.

Whitney [1969; 1972] developed the mathematics for the kinematics of multi-degree-of-freedom manipulators and prosthetic arms, and solved the inverse problem by finding the inverse Jacobian matrix, $\underline{J}^{-1}$, relating gripper motion to joint rate (i.e. $\dot{\underline{\theta}} = \underline{J}^{-1}\,\dot{\underline{X}}$). This method allows relative cartesian motion along world coordinates, based on real-time calculations of instantaneous $\underline{J}(\theta)$ and $\underline{J}^{-1}(\theta)$.

No general solution has been found for the inverse problem for absolute positions. An iterative solution based on finite displacement analysis has been proposed by Konstantinov [Konstantinov and Markov 1980; Konstantinov, Genova and Zahariev 1981; Konstantinov, Markov and Nechev 1981; Konstantinov and Patarinski 1982].

Several general recursive techniques have been proposed based on the approximate decoupling of the minor and major axis of motion [Gaglio et al 1981; Milenkov and Huang 1983; Benati et al 1982].

Explicit solutions can be found for most robot configurations of practical importance. Lee and Ziegler [1983] have derived an explicit solution for the PUMA robot based on geometry. Explicit solutions can also be obtained by matrix algebra using the Denavit-Hartenberg matrices [Paul, Shimano and Mayer 1981; Paul 1981].

Redundant degrees of freedom are useful for avoiding joint contraints and allowing collision-free configurations to be found. The inverse problem can be solved by optimizing some criterion function. Redundant systems have been considered in more detail by a number of investigators [Fournier and Khalil 1977; Liégeois 1977; Konstantinov Markov and Nechev 1981; Benati, Morasso and Tagliasco 1982].

We will develop an explicit inverse kinematics solution based on the matrix method of Paul [Paul, Shimano and Mayer 1981; Paul 1981].

5.2.1 Definition of Robot Joint Frames and Resulting Transformations

A robot arm consists of a series of rigid links connected by revolute or prismatic joints. Coordinate frames are attached to each link according to the specification of Denavit and Hartenberg [1955]. The geometry is shown in Figs. 3.6 and 3.7. First let us state some definitions:

$\underline{J}_i$ = joint i axis of rotation or translation

$\underline{a}_i$ = common normal of axes $\underline{J}_i$, $\underline{J}_{i+1}$.

For a revolute joint, the origin of F_i is located at the intersection of $\underline{J}_{i+1}$ and $\underline{a}_i$. If $\underline{J}_i$, $\underline{J}_{i+1}$ intersect, the origin is at the intersection point. If $\underline{J}_i$, $\underline{J}_{i+1}$ are parallel, $\underline{a}_i$ is chosen to intersect $\underline{J}_{i+1}$ at the same place as $\underline{a}_{i+1}$. If joint i+1 is prismatic, replace $\underline{J}_{i+1}$ above with the axis of the next revolute joint. The origin of F_i is defined for $d_{i+1} = 0$.

The z axis, $\underline{z}_i$, of F_i is along $\underline{J}_{i+1}$. The x-axis, $\underline{x}_i$, of F_i is along $\underline{a}_i$. If $\underline{J}_i$, $\underline{J}_{i+1}$ intersect, $\underline{x}_i$ is parallel or anti-parallel to $\underline{J}_i \times \underline{J}_{i+1}$. θ_i is zero when $\underline{x}_{i-1} = \underline{x}_i$.

For a prismatic joint, d_i becomes the joint variable. In this case $\underline{a}_i = 0$. The origin of F_i is coincident with the origin of F_{i-1} for $d_i = 0$. $\underline{z}_i$ is along $\underline{J}_{i+1}$ and $\underline{x}_i$ is parallel or anti-parallel to $\underline{J}_i \times \underline{J}_{i+1}$.

5.2.2 Inverse Kinematics Solution

5.2.2.1 General Approach to Inverse Kinematics Problem

Once we have defined the joint frames as shown, we can calculate the homogeneous transformation relating frame F_i to frame F_{i-1}. The transformation is composed of the following sequence:

1. rotate about z axis of F_{i-1} by an angle θ_i;
2. translate along z axis of F_{i-1} by distance d_i;
3. translate along new x axis of F_{i-1} by length a_i; and
4. rotate about new x axis of F_{i-1} by the twist angle α_i.

These four transformations can be combined as follows:

$$\underline{H}_{i-1,i} = \underline{\mathrm{Rot}}\ (z,\theta_i)\ \underline{\mathrm{Trans}}\ (z,d_i)\ \underline{\mathrm{Trans}}\ (x,a_i)\ \underline{\mathrm{Rot}}\ (x,\alpha_i)\ .$$

$$\underline{H}_{i-1,i} = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5.1)$$

where $C\theta_i \equiv \cos\theta_i$, $S\theta_i \equiv \sin\theta_i$, etc. Multiplying out, we get

$$\underline{H}_{i-1,i} = \begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} . \qquad (5.2)$$

This transformation was first derived by Denavit and Hartenberg [1955].

Once we have defined $\underline{H}_{i-1,i}$ for the links i=1, ... n, we can combine them to find each link frame relative to the robot base frame F_R, i.e.,

$$\underline{H}_{R,i} = \underline{H}_{R,1}\, \underline{H}_{1,2} \cdots \underline{H}_{i-1,i} \qquad (5.3)$$

for i = 1, n. We can also find the transformation

$$\underline{H}_{i,n} = \underline{H}_{i,i+1}, \cdots \underline{H}_{n-1,n} \qquad (5.4)$$

relating the last frame to the ith frame.

We now wish to solve for the joint variables for a given tool position and orientation. We can find the homogeneous transformation $\underline{H}_{R,n}$ which specifies the tool orientation and position as a function of the joint variables θ_1, θ_2, ... θ_n. We know the tool location and orientation relative to the robot base, $\underline{H}_{R,G}$. The tool frame is related to the last joint frame by a fixed known transformation $\underline{H}_{n,G}$, so we can easily find $\underline{H}_{R,n} = \underline{H}_{R,G}\, \underline{H}^{-1}_{n,G}$. The elements of $\underline{H}_{R,n}$ are now known: i.e.,

$$\underline{H}_{R,n} = [\ \underline{n}, \underline{o}, \underline{a}, \underline{P}\ ] \ , \qquad (5.5)$$

where

$\underline{n}$, $\underline{o}$, $\underline{a}$ are basis vectors of link n frame relative to F_R;

$\underline{P}$ is position vector of link n frame relative to F_R.

Let us call this matrix $\underline{T}_n$.

We can now equate $\underline{T}_n$ and $\underline{H}_{R,n}$ element by element, searching for explicit equations for the joint angles $\theta_1, \cdots \theta_n$. If necessary, we can also compare the elements of other equivalent matrices defined below.

$$\underline{T}_n = \underline{H}_{R,n} \quad , \tag{5.6}$$

$$\underline{H}_{R,1}^{-1} \underline{T}_n = \underline{H}_{1,n} \quad , \tag{5.7}$$

$$\underline{H}_{1,2}^{-1} \underline{H}_{R,1}^{-1} \underline{T}_n = \underline{H}_{2,n} \quad , \text{ etc.} \tag{5.8}$$

We continue until we have explicit solutions for all the joint variables.

The actual solution of a particular robot is generally based on intuition and simplification. The particular solution for a UNIMATION PUMA 560 follows.

5.2.2.2 PUMA 560 Solution

For the PUMA robot, we can define a set of link frames as shown in Fig. 5.1. We can also specify the appropriate link parameters in Table 5.1.

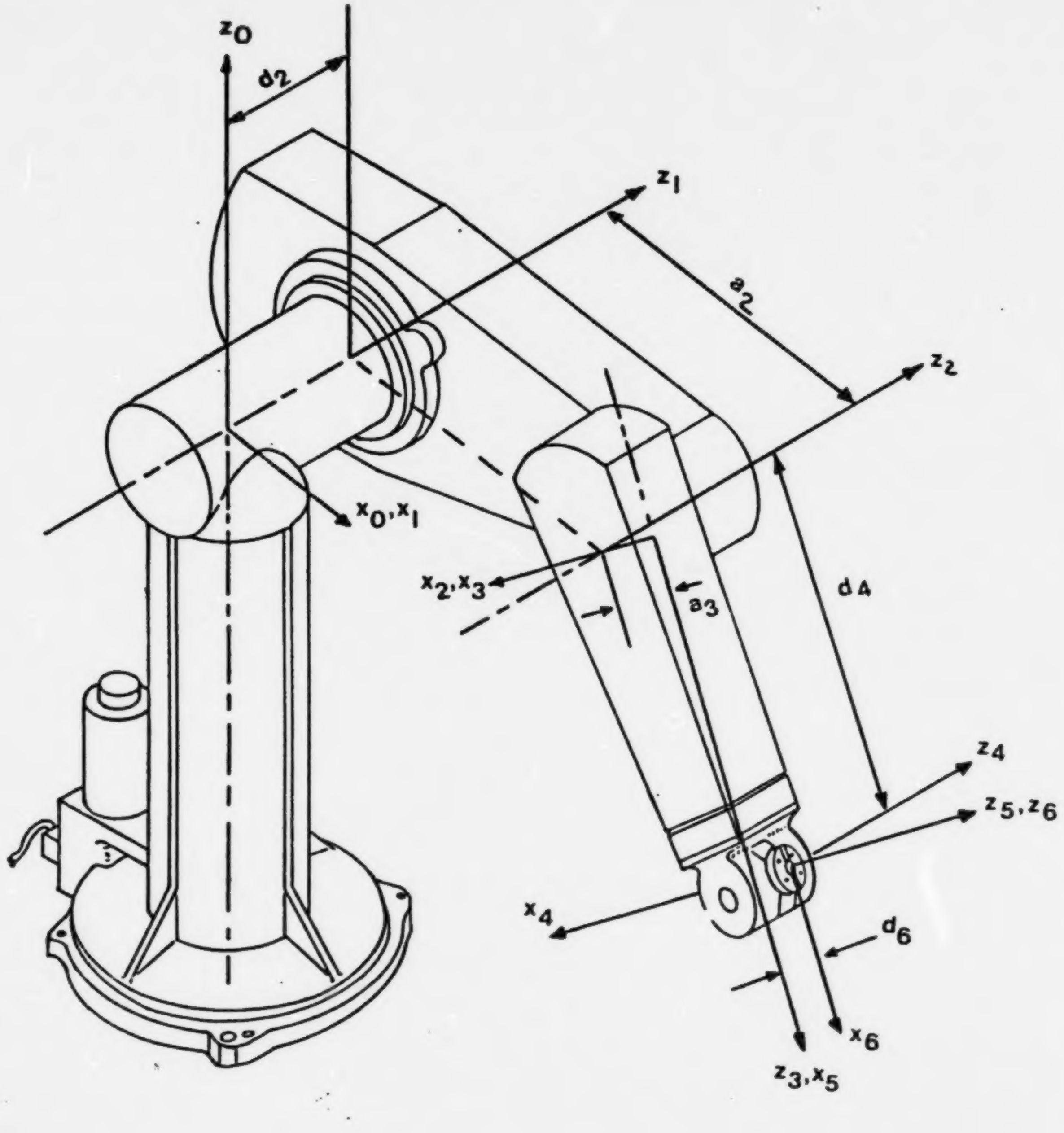

z_0
d_2
z_1
a_2
z_2
x_0, x_1
x_2, x_3
a_3
d_4
z_4
z_5, z_6
x_4
d_6
x_6
z_3, x_5

Figure 5-1. PUMA Joint Frames and Link Parameters.

Table 5.1 - PUMA Link Parameters

Link	Variable	α	a	d
1	θ_1	-90	0	0
2	θ_2	0	a_2	d_2
3	θ_3	90	$-a_3$	0
4	θ_4	-90	0	d_4
5	θ_5	90	0	0
6	θ_6	0	0	d_6

Note that the sign of a_3 is negative to conform to our definition of x_2. For the PUMA, the link parameters have the following values:

$a_2 = 432$ mm

$a_3 = 20.5$ mm

$d_2 = 149.5$ mm

$d_4 = 432$ mm

$d_6 = 56.5$ mm

Note that the following analysis also applies to any other 6 degrees of freedom robot with the same geometry. The correct values of the link parameters a_2, a_3, d_2, d_4, and d_6 are simply substituted into the solutions.

We know the position and orientation of F_6 relative to F_0 from

$$\underline{H}_{0,6} = [\ \underline{n},\ \underline{o},\ \underline{a},\ \underline{P}\ ] \quad , \tag{5.9}$$

where

$\underline{n}, \underline{o}, \underline{a}$ are the F_6 basis vectors projected onto F_0;

$\underline{P}$ is the F_6 origin relative to F_0 .

Note that F_0 is the frame attached to link 0 by our definition, and is equivalent to the robot base frame F_R. F_6 is attached to the tool mounting flange.

We have defined the inter-link transformations $\underline{H}_{i-1,i}$ for each link and we know that $\underline{H}_{0,i} = \underline{H}_{0,1}\,\underline{H}_{1,2} \cdots \underline{H}_{i-1,i}$. (5.10)

From the geometry of the arm (Figure 5.2) we note that the origin of F_4 is dependent on the variables θ_1, θ_2, θ_3 only, and is coincident with the origin of F_5.

Let $\underline{q} = \underline{P} - d_6\,\underline{a}$ (origin of F_5) , (5.11)

and $\underline{q}$ = 4^{th} column of $\underline{H}_{0,4}$ (origin of F_4) .

We can find $\underline{q}$ by multiplying $\underline{H}_{0,1}\,\underline{H}_{1,2}\,\underline{H}_{2,3}\,\underline{H}_{3,4}$ (we need the 4^{th} column only).

The 4th column of $\underline{H}_{2,4}$ is $\begin{bmatrix} S_3d_4 + a_3C_3 \\ -C_3d_4 + a_3S_3 \\ 0 \\ 1 \end{bmatrix}$.

The 4th column of $\underline{H}_{1,4}$ is $\begin{bmatrix} C_2(S_3d_4 + a_3C_3) + S_2(C_3d_4 - a_3S_3) + a_2C_2 \\ S_2(S_3d_4 + a_3C_3) - C_2(C_3d_4 - a_3S_3) + a_2S_2 \\ d_2 \\ 1 \end{bmatrix}$.

$\underline{q}$ = 4th column of $\underline{H}_{0,4}$.

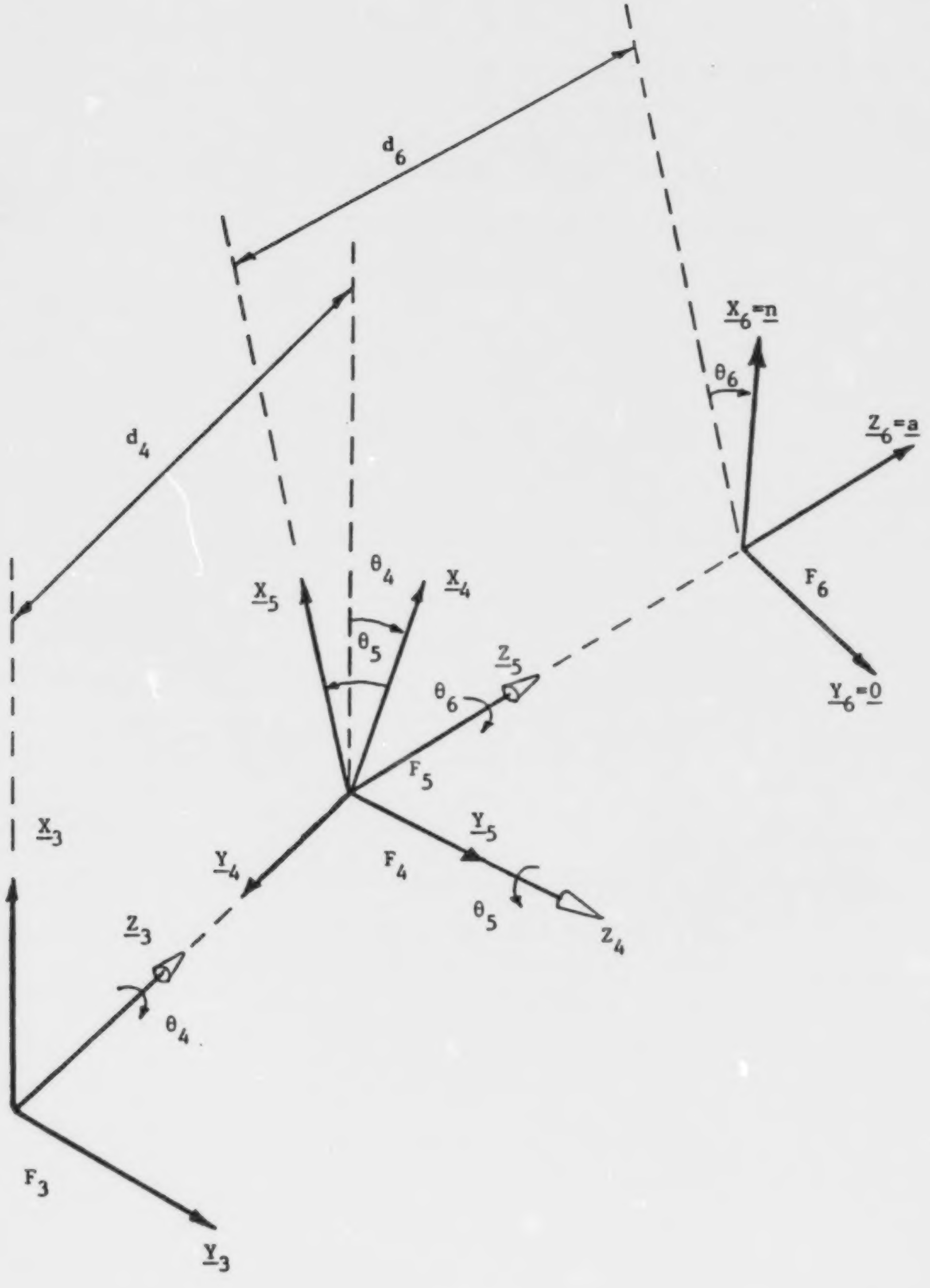

d_6
d_4
$\underline{X}_6 = \underline{n}$
θ_6
$\underline{Z}_6 = \underline{a}$
F_6
$\underline{Y}_6 = \underline{0}$
θ_4
$\underline{X}_5$
$\underline{X}_4$
θ_5
$\underline{Z}_5$
θ_6
F_5
$\underline{X}_3$
$\underline{Y}_5$
$\underline{Y}_4$
F_4
θ_5
Z_4
$\underline{Z}_3$
θ_4
F_3
$\underline{Y}_3$

Figure 5-2. Relative geometry of PUMA link frames F_3, F_4, F_5, F_6.

Multiplying out, we get

$$\underline{q} = \begin{bmatrix} C_1[C_2(S_2d_4 + a_3C_3) + S_2(C_3d_4 - a_3S_3) + a_2C_2] - S_1d_2 \\ S_1[C_2(S_3d_4 + a_3C_3) + S_2(C_3d_4 - a_3S_3) + a_2C_2] + C_1d_2 \\ -S_2(S_3d_4 + a_3C_3) + C_2(C_3d_4 - a_3S_3) - a_2S_2 \\ 1 \end{bmatrix} . \quad (5.12)$$

$\underline{q}$ can be simplified as follows:

$$\underline{q} = \begin{bmatrix} C_1[d_4S_{23} + a_3C_{23} + a_2C_2] - S_1d_2 \\ S_1[d_4S_{23} + a_3C_{23} + a_2C_2] + C_1d_2 \\ d_4C_{23} - a_3S_{23} - a_2S_2 \\ 1 \end{bmatrix} , \quad (5.13)$$

where

$$S_{23} = \sin(\theta_2 + \theta_3)$$
$$C_{23} = \cos(\theta_2 + \theta_3) \quad .$$

Solution for θ_1

From the first two rows of (5.13) we can show that

$$\theta_1 = \tan^{-1} \left[\frac{\pm q_2(q_1^2 + q_2^2 - d_2^2)^{1/2} - d_2 q_1}{\pm q_1(q_1^2 + q_2^2 - d_2^2)^{1/2} + d_2q_2} \right] , \quad (5.14)$$

+ for left handed arm configuration,

- for right handed arm configuration.

Note: For $\theta = \tan^{-1}\left(\frac{x}{y}\right)$;

$0 < \theta < 90°$ for $x > 0$ and $y > 0$;

$90° < \theta < 180°$ for $x < 0$ and $y > 0$;

$-180° < \theta < -90°$ for $x < 0$ and $y < 0$;

$-90° < \theta < 0$ for $x > 0$ and $y < 0$.

Solution for θ_3

Let

$$A = \frac{q_1^2 + q_2^2 + q_3^2 - d_2^2 - d_4^2 - a_3^2 - a_2^2}{2\ a_2} ; \qquad (5.15)$$

$$B = -d_4 \ ; \qquad C = a_3 \ . \qquad (5.16),(5.17)$$

We can show that $A = -B \sin \theta_3 + C \cos \theta_3$. The solution has the following form:

$$\theta_3 = \tan^{-1}\left(\frac{C}{B}\right) - \tan^{-1}\left(\frac{A}{\pm\sqrt{r^2 - A^2}}\right) \qquad (5.18)$$

where

$$r^2 = B^2 + C^2 \ . \qquad (5.19)$$

Substituting (5.16), (5.17) into (5.18) we get

$$\theta_3 = \tan^{-1}\left(\frac{a_3}{-d_4}\right) - \tan^{-1}\left(\frac{A}{\pm\sqrt{r^2 - A^2}}\right) \qquad (5.20)$$

where $$r^2 = a_3^2 + d_4^2 \ . \qquad (5.21)$$

Two different solutions result depending on the sign:

+ for elbow above configuration,

and – for elbow below configuration, (for left arm configuration);

or - for elbow above configuration,

and + for elbow below configuration, (for right arm configuration).

Solution for θ_2

From (5.13) we get

$$q_3 = d_4C_{23} - a_3S_{23} - a_2S_2 \quad . \tag{5.22}$$

Expanding, we get

$$q_3 = d_4[C_2C_3 - S_2S_3] - a_3[S_2C_3 + C_2S_3] - a_2s_2 \; . \tag{5.23}$$

Rearranging we get

$$q_3 = [d_4C_3 - a_3S_3]\,C_2 - [d_4S_3 + a_3C_3 + a_2]\,S_2 \; . \tag{5.24}$$

Let

$$A = d_4C_3 - a_3S_3 \; , \tag{5.25}$$

$$B = d_4s_3 + a_3C_3 + a_2 \quad . \tag{5.26}$$

Then the solution is

$$\theta_2 = \tan^{-1}\left(\frac{A}{B}\right) - \tan^{-1}\left(\frac{q_3}{\pm\sqrt{r^2 - q_3^2}}\right) \quad . \tag{5.27}$$

Substituting (5.25), (5.26) into (5.27) we get

$$\theta_2 = \tan^{-1}\left(\frac{d_4C_3 - a_3S_3}{d_4S_3 + a_3C_3 + a_2}\right) - \tan^{-1}\left(\frac{q_3}{\pm\sqrt{r^2 - q_3^2}}\right) \tag{5.28}$$

where

$$r^2 = A^2 + B^2 \quad . \tag{5.29}$$

Expanding (5.29), we get

$$r^2 = (d_4C_3 - a_3S_3)^2 + (d_4S_3 + a_3C_3 + a_2)^2 \quad . \tag{5.30}$$

Again, there are two solutions for (5.28);

+ is for the left handed arm configuration

and − is for the right handed arm configuration.

There are two solutions for θ_2 because the parameters q_1 and q_2 are not in the equation. It is intuitively obvious that for a given waist rotation θ_1, there exist two possible shoulder rotations, θ_2, which position the arm at the same height q_3. The possible rotations position the arm in opposite quadrants of the horizontal plane. The correct solution can be determined by substituting θ_2 into the equation for the parameter q_1 or q_2 (equation 5.1) and comparing the sign of the result. It turns out, however, that the positive solution is always associated with the left arm configuration and the negative solution with the right arm configuration.

Solution for θ_4, θ_5, θ_6

Set θ_4 such that $\underline{z}_4$ is normal to the plane formed by $\underline{z}_3$ and $\underline{a}$, i.e.,

$$\underline{z}_4 = \pm \frac{(\underline{z}_3 \times \underline{a})}{|\underline{z}_3 \times \underline{a}|} \tag{5.31}$$

and

$$\sin \theta_4 = \pm \left(\underline{x}_3 \cdot \underline{z}_4\right) , \tag{5.32}$$

$$\cos \theta_4 = \pm \left(\underline{y}_3 \cdot \underline{z}_4\right) . \tag{5.33}$$

Note that $\underline{x}_3$, $\underline{y}_3$ are the first two columns of $\underline{H}_{0,3}$. Expanding out $\underline{H}_{0,3}$ and substituting $\underline{x}_3$, $\underline{y}_3$ and $\underline{z}_4$ into (5.32), (5.33), we can show

$$\theta_4 = \tan^{-1} \left[\frac{\pm(C_1 a_y - S_1 a_x)}{\pm (C_1 C_{23} a_x + S_1 C_{23} a_y - S_{23} a_z)} \right] \tag{5.34}$$

where + gives wrist not flipped configuration

and − gives wrist flipped configuration.

Set θ_5 so that $\underline{z}_5 = \underline{a}$, (5.35)

i.e., $\sin\theta_5 = \underline{x}_4 \cdot \underline{a}$, (5.36)

and $\cos\theta_5 = -\underline{y}_4 \cdot \underline{a}$. (5.37)

$\underline{x}_4$, $\underline{y}_4$ are the first two columns of $\underline{H}_{0,4}$. Expanding out $\underline{H}_{0,4}$ and substituting $\underline{x}_4$, $\underline{y}_4$ into (5.36), (5.37) we can show

$$\theta_5 = \tan^{-1}\left[\frac{(C_1C_{23}C_4 - S_1S_4)a_x + (S_1C_{23}C_4 + C_1S_4)\,a_y - C_4S_{23}a_z}{C_1S_{23}a_x + S_1S_{23}a_y + C_{23}a_z} \right] . \quad (5.38)$$

Set θ_6 so that $\underline{y}_6 = \underline{o}$, (5.39)

i.e., $\sin\theta_6 = \underline{y}_5 \cdot \underline{n}$, (5.40)

and $\cos\theta_6 = \underline{y}_5 \cdot \underline{o}$. (5.41)

$\underline{y}_5$ is the second column of $\underline{H}_{0,5}$.

Multiplying out $\underline{H}_{0,5}$ and substituting $\underline{y}_5$ into (5.40), (5.41) we can show

$$\theta_6 = \tan^{-1}\left[\frac{(-S_1C_4-C_1C_{23}S_4)n_x + (C_1C_4 - S_1C_{23}S_4)n_y + S_4S_{23}n_z}{(-S_1C_4-C_1C_{23}S_4)o_x + (C_1C_4-S_1C_{23}S_4)o_y + S_4S_{23}o_z} \right] . \quad (5.42)$$

5.2.2.3 Discussion of PUMA Solutions

When we inspect the joint angle solutions, we note that joints 1, 3 and 4 have 2 solutions each. These solutions correspond to different robot configurations which yield the same tool position $\underline{H}_{0,6}$. For instance, the arm may be in a left-handed or right-handed configuration, the elbow may be above or below the tool, and the wrist may be flipped. This suggests that different configurations might be tried when searching for a feasible configuration. In order to obtain smooth motion between points, however, we must insist that the same configuration be maintained throughout a given trajectory.

5.2.3 Comparison of Five Degree of Freedom and Six Degree of Freedom Robots

A 6 degree of freedom robot allows us to completely specify the tool position and orientation in space. In the case of welding, however, we note that the tool is symmetric about its axis. This means that any rotation about the tool axis does not change the effective tool orientation. Thus we see that we need only specify the tool position and a single orientation vector $\underline{a}$. Five degrees of freedom are sufficient and a 6th degree of freedom is redundant. For a 5 degree of freedom robot, the tool frame basis vectors $\underline{n}$, $\underline{o}$ are found to be functions of the orientation vector $\underline{a}$ and the position vector $\underline{P}$. The relationships are derived in the following analysis for a PUMA robot with the 4th joint omitted.

If we have 6 degrees of freedom, we can independently define the position $\underline{P}$ and any 2 of the basis vectors $\underline{n}$, $\underline{o}$, $\underline{a}$ (the third basis vector is defined by the other 2). This means that we can explicitly rotate F_6 about the torch axis $\underline{a}$ without changing the tools effective orientation (Fig. 5.3). We will use this trick in searching for feasible solutions of a trajectory by specifying a rotation of F_6 about $\underline{a}$, calculating a new $\underline{H}_{0,6}$ and testing the resulting joint angles for feasibility.

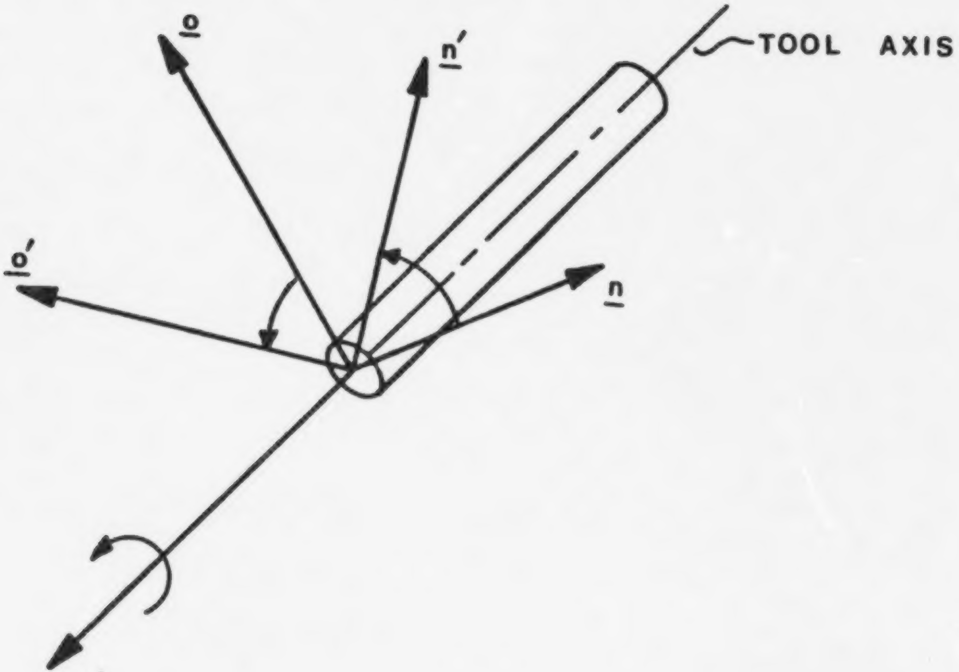

o
n′
TOOL AXIS
o′
n
a, a′

Figure 5-3. Rotation of Tool Frame About Tool Axis.

5.3 Interference Avoidance

The interference avoidance problem has been well summarized by Lozano-Perez [Brady et al 1982]. The three classes of interference avoidance algorithm are: hypothesize and test, penalty functions and explicit free space.

The hypothesize and test algorithm generates a collision free path by trial and error. Each time a collision is detected, an alternate path is formulated and tested. This method is simple but does not lead to an optimum solution. A solution may not be found at all because the knowledge of the world is localized and thus a global strategy is impossible.

The penalty function scheme attempts to define a penalty function whose value reflects the probability of collision as a function of position in space. The algorithm then generates a path following the local minima of the penalty function. This method has been successfully applied to extremely simple world models only, and the algorithm becomes very complex for more realistic situations.

The third class of algorithm generates a set of all robot positions which are free of collisions and calls this set the free space. The shortest path through the free space joining an initial and final position is the optimum path. The main difficulty with this method lies in generating the free space, and all algorithms to date incorporate simplifications and approximations which restrict their generality. Several free space path planning algorithms have been presented in the literature [Udupa 1977], [Lozano-Perez and Wesley 1979], [Lozano-Perez 1983], [Brooks 1983].

For simplicity we have chosen to explore the hypothesis and test approach.

6. INTERFERENCE DETECTION

Detection of interference between solid objects is a fundamental part of the interference avoidance problem.

The simplest problem is the case of interference between fixed, stationary solids. Hosaka [1974] describes a method for completely specifying the intersection between two stationary convex polyhedra. Maruyama [1972] has proposed a simple method for determining intersections between polyhedral solids using a hierarchy of tests.

The problem of interference between moving and stationary polyhedra has been considered by Boyse [1979]. The tests that he devised apply only to the simple cases of pure translation and rotation and cannot be extended to the more general case of complicated motion. An algorithm for testing for interference between two moving convex polyhedra is suggested by Schwartz [1981]. His method finds the minimum distance between two convex polyhedra as a function of time if their individual trajectories are known functions of time. Schwartz demonstrates the algorithm for the two dimensional case only.

Another way of handling interference detection between moving objects is to define a swept volume generated by the moving object. The problem is then reduced to the simpler case of interference between stationary solids. This method is discussed by Requicha [1980] and Lozano-Perez and Wesley [1979].

For simplicity, we have chosen to consider the problem of interference between stationary solids only. The problem of moving solids can be easily approximated by a series of stationary problems. For accuracy, the step size between stationary positions must be as small as possible, but this increases the number of cases to be tested. It is therefore important that the interference detection algorithm be as fast and efficient as possible.

Most previous work described in the literature is concerned with the case of intersecting polyhedra. The simplest and most obvious algorithm [Boyse, 79] tests for intersection of any edge of one polyhedron with any face of the other. Detection of intersection is simple, but determining the volume of intersection is more difficult. More sophisticated methods for polyhedral intersection testing have been proposed [Maruyama, 72], [Hosaka, 74], [Braid, 75], [Kalay, 81], [Franklin, 82].

6.1 Face Splitting Method for Interference Detection

Many of the operations performed in hidden surface removal are similar to operations required for interference detection. This leads to the idea of extending the Newell/Newell/Sancha (N/N/S) priority list hidden surface algorithm to include interference detection and depiction.

The N/N/S list priority hidden surface algorithm is well described in the literature [Rogers,85], [Foley and Van Dam,82]. In essence, the algorithm sorts the faces of the polyhedra in the scene so that the faces furthest from the viewpoint are output first. Closer faces automatically obscure more distant faces, effectively removing hidden surfaces. A hierarchy of tests is performed on each face to determine if it might obscure any other face. If not, the face is output on the screen, and is no longer considered. This process is repeated until all the faces have been output.

We can incorporate interference detection and depiction of the volume of intersection by making a few simple changes to the algorithm:

1. The N/N/S algorithm initially eliminates all self-hidden faces because they are never visible. Our algorithm will consider ALL of the faces.
2. The N/N/S algorithm splits faces only as a last resort when all other tests fail. We will split all intersecting faces so that no face in the final list intersects any of the original unsplit faces. In other words, whenever a pair of faces intersect, BOTH faces are split along their line of intersection.

As a result of these changes, we end up with more face fragments in the final list than we need for hidden surface removal. However, we have a complete set of faces from which we can construct the union or intersection of the two polyhedra. The intersection is the volume of interference. As an added bonus, the faces are already ordered by output priority.

6.1.1 A Hierarchy of Tests. The modified N/N/S algorithm is composed of the following hierarchy of tests:

6.1.1.1 Preliminary Depth Sort All faces are sorted by maximum depth into the screen (maximum Z). The deepest faces are at the front of the list.

6.1.1.2 Overlap of X-Y Bounding Boxes The bounding boxes of all faces Q which overlap the Z extent of the current candidate face P are tested against the bounding box of P.

6.1.1.3 Face Plane Tests The face P is tested against the faces Q to see if P is wholly behind the plane of Q AND if Q is wholly in front of the plane of P.

6.1.1.4 Screen Overlap Test Faces which possibly intersect are tested for overlap in screen space.

6.1.2 The Proposed Algorithm The algorithm is described as follows:

1. Sort the face list by maximum screen depth and keep it sorted. Proceed through the following hierarchy of tests until all faces have been ordered.
    a. Find the set of faces Q which overlap in depth the face p currently at the front of the face list.
    b. Eliminate the faces Q whose X-Y bounding boxes do not overlap the X-Y bounding box of P.
    c. Eliminate the remaining faces Q for which the following is true: P is wholly on the far side of the plane of Q AND Q is wholly on the near side of the plane of P.
    d. Eliminate the remaining faces Q which do not overlap P in screen space.
2. If no faces Q remain, then add P to the output face list and remove P from the current face list.
3. Otherwise, for every Q, test if Q is wholly on the far side of the plane of P and if P is wholly on the near side of the plane of Q.
4. If so, then replace P with a face Q and insert P into the face list according to depth. Return to the start and repeat.

5. Otherwise, split the faces P and Q as necessary. Remove the old faces from the list and insert the new fragments according to depth. Return to the start and repeat.

This procedure is repeated until no faces are left in the current face list. The result is a priority sorted list of all possible face fragments, which can be output directly if desired.

6.1.3 Finding the Volume of Intersection. A face fragment is a face of the volume of intersection if it meets one of the following conditions:

1. It is interior to one of the polyhedra.
2. It lies on the face of one of the polyhedra and its surface normal matches the polyhedron face normal in sign.

For convex polyhedra, these tests are simple. A face fragment lies inside a polyhedron if it lies "inside" all of the face planes of the polyhedron. We have eliminated all intersecting faces, so we need only test a single vertex of a fragment which does not lie on the face plane. If all of a fragment's vertices lie on the face, then the fragment and face are coincident. If their surface normals match in sign, then the fragment is a face of the intersection volume. Otherwise, the fragment represents contact between two disjoint polyhedra. The algorithm can be stated as follows:

1. Find all face fragments of polyhedron 2 which are interior to all of the original faces of polyhedron 1.
2. Find all face fragments of polyhedron 1 which are interior to all of the fragments found above.

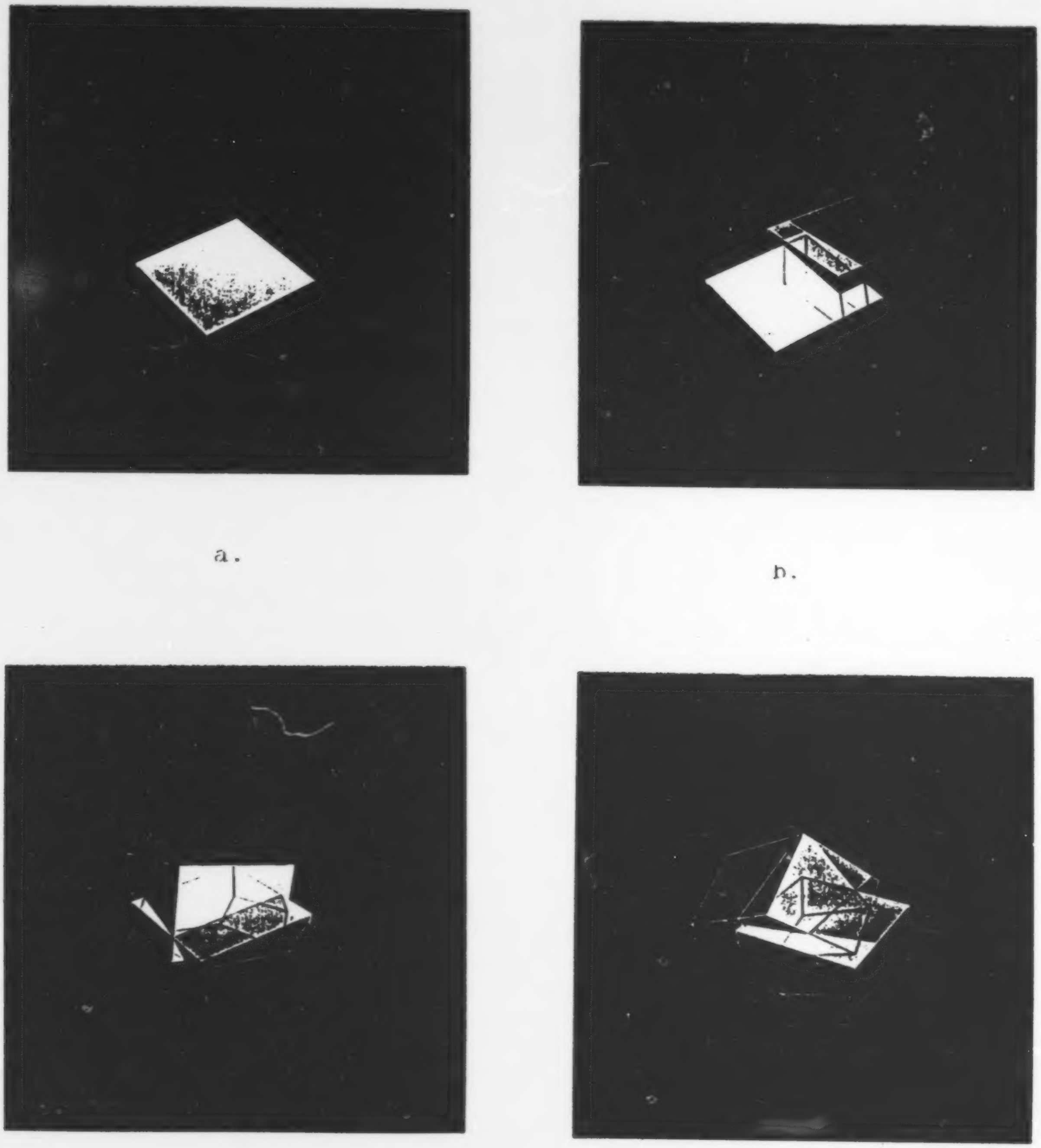
We now have a priority ordered list of faces for the volume of intersection. We have many options in how we can depict the intersection. For example, we can superimpose a wire-frame image, we can superimpose a solid image in some other colour, or we can depict the intersection volume by itself.

6.1.4 Implementation and Evaluation. The algorithm described above was implemented in FORTRAN and tested on the IRIS at an early stage of the contract period. The data structures and program listings are shown in Appendix I. The data structures have subsequently been slightly modified.

Two identical cubes were modelled for testing purposes. The test program allows the user to interactively change the viewpoint. One of the cubes can also be interactively moved in world coordinates relative to the other. The hidden surface removal and interference detection algorithm is performed each time a new frame is shown. An elementary shading algorithm is performed on each face before it is output on the screen. If interference occurs, the interference volume is shown as a superimposed wire-frame representation. Figures 6.1a - 6.1d show some typical output of the program.

A number of comments can be made about the initial impl[illegible]ation:

1. The cubes appear very fractured when interference occurs, because each face in the output list is surrounded by a white line for visibility. As a result, all lines of intersection are shown. Also, a perspective projection is used, so the dot product of the line of sight and the surface normal varies across a surface. Thus,



a.

b.

c.

d.

Figure 6-1. Photographs of Screen Output

different constant shades are chosen for different fragments of the same face, resulting in discontinuous shading. These problems only occur when there is intersection, and are not usually noticeable if the objects are spatially separated.

2. The algorithm is fast enough for real time response (for two cubes at least) only when the objects are disjoint. For a scene such as in Fig. 6.1a, the screen is updated at least once a second. The intersection calculations slow the process down considerably. For complicated intersections such as shown in Figs. 6.1b, 6.1c and 6.1d, the program takes about 10 to 15 seconds to update the screen.

   Objects will normally be disjoint, so the algorithm is useful for interactive graphics despite the slow execution of the intersection calculations.

3. The geometric complexity increases greatly when intersection occurs. A comparison of the scenes in Photos 6.1a and 6.1d is shown in Table 6.1 to illustrate this point.

6.1.5 Some Possible Improvements. The software was not designed with efficiency as a primary goal, and considerable optimization is possible. Some likely areas for improvement include: better list sorting algorithms, elimination of redundant calculations, and streamlining of the data structure. A detailed analysis is required to identify specific bottlenecks in the program.

An increase in speed can be realized if the interference test stops as soon as interference is detected, without generating the interference volume. This can be incorporated as a user selected option. Some modifications to the algorithm and possibly some additional tests are necessary so that intersection is detected quickly.

TABLE 6-1. Comparison of Two Test Cases

	Fig. 6.1a	Fig. 6.1d
Total number of faces	12	68
Total number of vertex pointers	48	275
Total number of vertices	16	72
Number of faces output	12	40

Hidden surface removal for disjoint objects can be accelerated by implementing another level in the algorithm hierarchy. For convex polyhedra which are disjoint, it is sufficient to sort the polyhedra by depth, and to output them as single entities. This is analogous to the N/N/S sorting procedure except that polyhedra are sorted rather than faces. Face sorting need only be initiated when the polyhedral sort fails.

6.2 Two Interference Methods Developed by J.P. Duncan

Two interference detection methods have been developed by J.P. Duncan as a contribution to this project. His methods are described fully in Appendix II. The two methods are summarized below.

1. Method I

The first method is based on a so-called tetrahedral concept. One of the two polyhedral solids to be tested is densely defined by a fine grid of surface points. Interference occurs if any of these surface points is interior to the second sparsely defined test polyhedron.

The sparsely defined polyhedron is specified such that all its faces are triangles. This introduces redundant information for most polyhedra. A tetrahedron can be defined by a facet and any point not in the same plane. A point is tested for inclusion in the sparsely defined polyhedron by generating the set of tetrahedra defined by the point and all of the facets. If the sum of their volumes equals the volume of the sparse polyhedron, then the point is interior. If the sum is greater, then the point is exterior. A fast method for determining the tetrahedral volumes has been developed by Duncan. Each surface point of the densely defined polyhedron can thus be tested against the sparsely defined polyhedron. If any are interior, then interference between the objects has occured.

This method requires that redundant points be added to the definition of one of the two objects if both are sparsely defined polyhedra. The densely defined polyhedron can be convex or concave. The sparsely defined polyhedron must be convex, or a member of a restricted set of concave polyhedra. The possiblility exists of the two polyhedra intersecting without detection.

This method can provide either a simple interference check, or it can generate a complete 'cloud' of intersecting points.

2. Method II

Duncan's second method is based on edge-face intersection. Two polyhedra intersect if any edge of one intersects any face of the other, or if one is wholly included by the other. Method II considers only the possibility of edge-face intersection, which is acceptable for detecting intersection after a small step position change.

An edge-face test is performed by first finding the intersection of the corresponding line and plane. If the intersection point lies on the edge, and within the face boundaries, then intersection has occured.

This method applies to both convex and concave polyhedra.

Both methods I and II compare the bounding boxes of the polyhedra in space as a preliminary test.

Duncan represents the topology of all polyhedra by a two-dimensional grid analogous to a "Mercator's Projection". Each grid point specifies a vertex of the polyhedron. Redundant points must be added to represent North and South poles. Unfortunately, a direct mapping does not exist between the vertices of most polyhedra and a Mercator's Projection. This forces the addition of redundant points to the polyhedral representation. These points are not vertices, but lie on the faces and edges. Fig. 6-4 shows an example of this. A more general representation of the topology is possible using a connected graph as shown in Fig. 6.5.

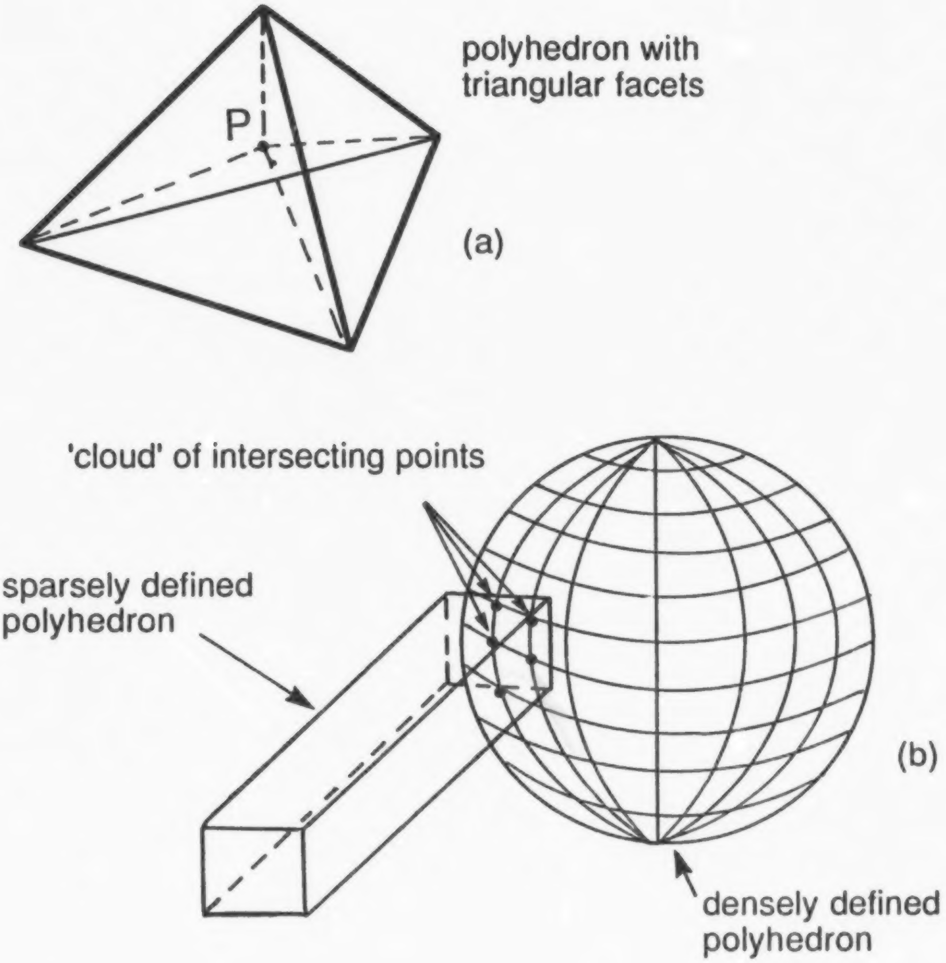

polyhedron with
triangular facets
P
(a)
'cloud' of intersecting points
sparsely defined
polyhedron
(b)
densely defined
polyhedron

Figure 6-2. Duncan Interference Method I.

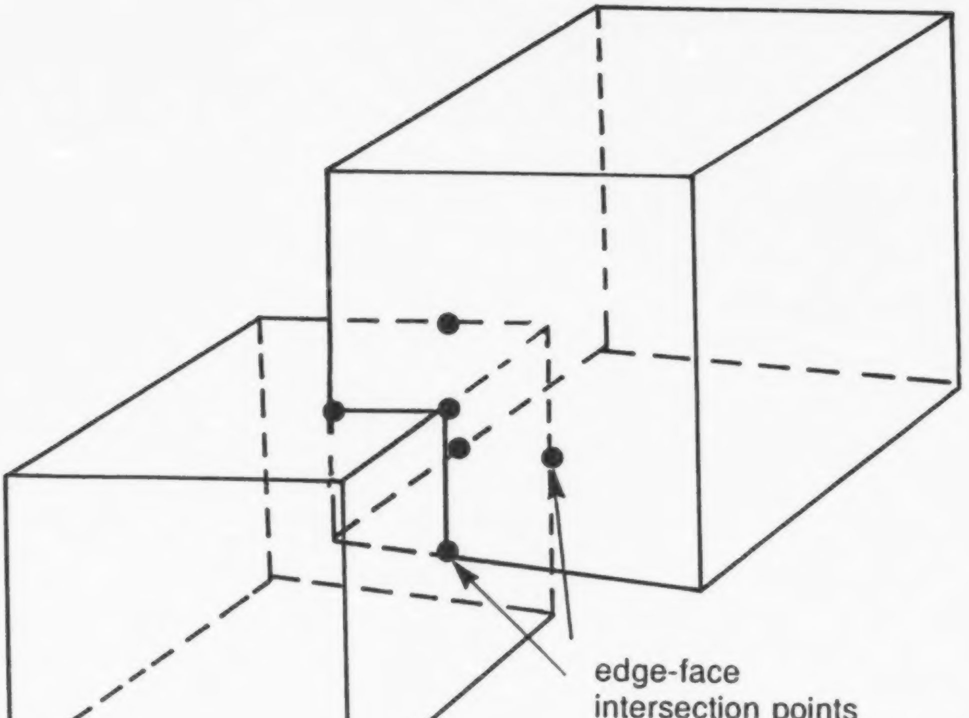

edge-face
intersection points

Figure 6-3. Duncan Interference Method II

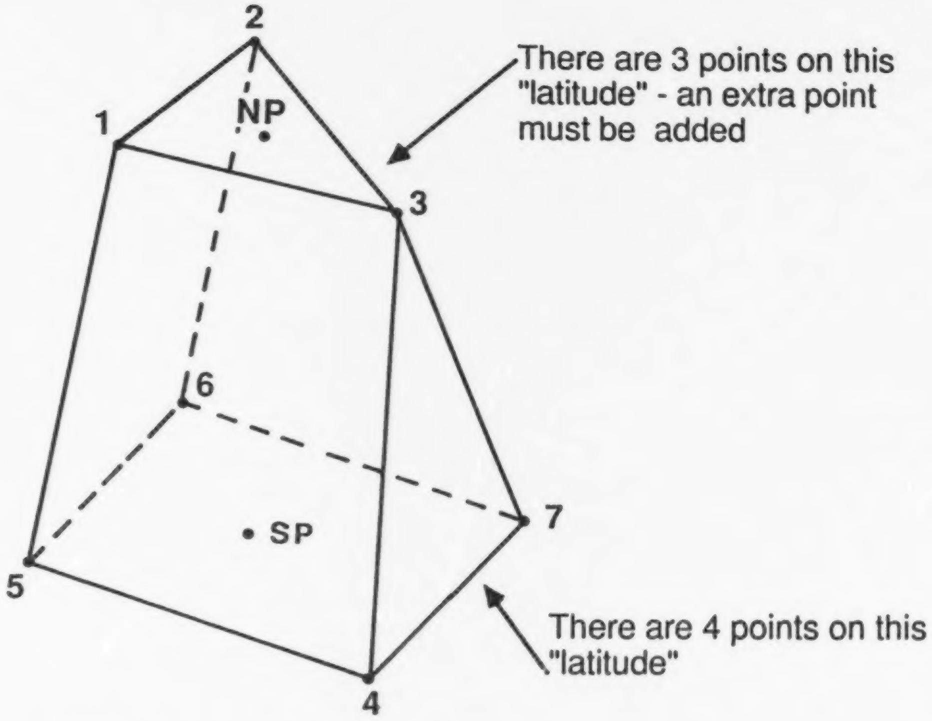
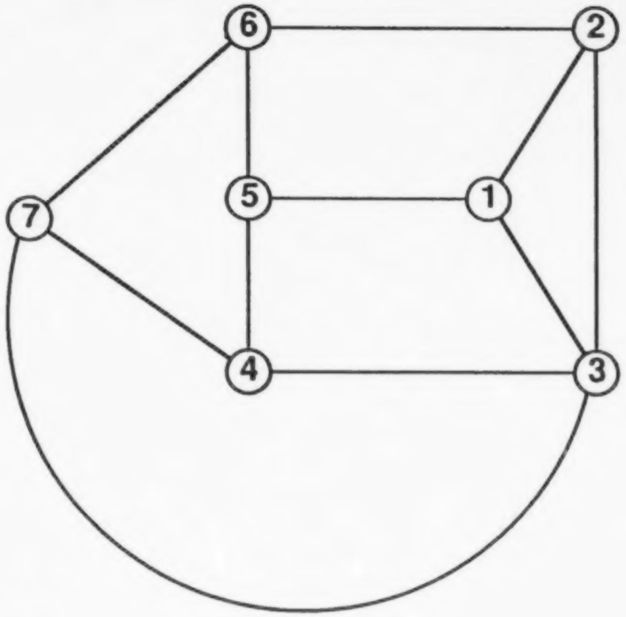

2
1
NP
3
6
7
SP
5
4
There are 3 points on this "latitude" - an extra point must be added
There are 4 points on this "latitude"

Figure 6-4. Example of a Polyhedron Not Represented by a Mercator's Projection.



node	adjacent nodes
1	2 3 5
2	1 3 6
3	1 2 4 7
4	3 5 7
5	1 4 6
6	2 5 7
7	3 4 6

Figure 6-5. A Connected Graph Representation.

The Duncan methods were developed and tested on a personal computer and were converted into Fortran to run on a VAX. The method I software has been transferred to the IRIS-1400, and successfully run.

6.3 Separating Plane Method for Interference Detection

This method provides a fast way to check for interference between convex polyhedral solids. If desired, the shortest separating distance and its direction can be found, at the expense of more processing and slower response.

If two convex polyhedral solids are disjoint in space, then there exists at least one plane which separates them into disjoint half-spaces. Conversely, if no such plane exists, then the two polyhedra intersect. Interference detection in a complex environment can be broken down into searching for separating planes between pairs of convex polyhedra. To eliminate unnecessary processing, pairs which are known to be disjoint due to kinematic constraints are not tested.

6.3.1 A Fast Search for a Separating Plane. A separating plane between two convex polyhedra can be found by a method analogous to the method of steepest descent for minimization problems. We will impose the requirement that the candidate plane P be tangent to one of the polyhedra, POLY1, and we will define its orientation such that its outward normal is directed away from POLY1. The function to be maximized is the minimum distance from P to the second polyhedron, POLY2. We will call this distance DMIN(P). The plane is a separating plane if DMIN(P) is positive. The optimum plane is the one which maximizes DMIN(P).

6.3.2 Best Separating Face. For most cases, at least one face of a pair of convex polyhedra defines a separating plane between them as illustrated by Fig. 6-6. A fast search of the faces F of each polyhedron against the other is performed. The search is along the path which most rapidly increases DMIN(F).

A face defines a separating plane if DMIN(F) is positive. If a face gives a larger DMIN(F) than any of its neighbours, then it is the best separating face of that polyhedron. If this search is repeated for the other polyhedron, then we can get the best separating face for the pair.

If the best separating face of a pair of polyhedra results in a negative DMIN(F), then no face defines a separating plane, although the polyhedra may be disjoint as shown in Fig. 6-7. We must now find the shortest separating vector connecting the two polyhedra. The optimum separating plane is normal to this vector, and is tangent to the first polyhedron. If the length of the separating vector, DMIN(P), is negative, then the two polyhedra intersect.

6.3.3 Frame Coherence. The scene usually changes only slightly from one graphical frame to the next. We call this frame coherence. If we store the separating face and closest vertex for each pair of parts, then we can use them as a starting point for the next separating face search. The search will generally be much shorter than if some arbitrary starting point is used each time.

6.3.4 Calculation of DMIN(P). DMIN(P) is the minimum distance from a plane P to a second polyhedron POLY2. Equivalently, DMIN(P) is the

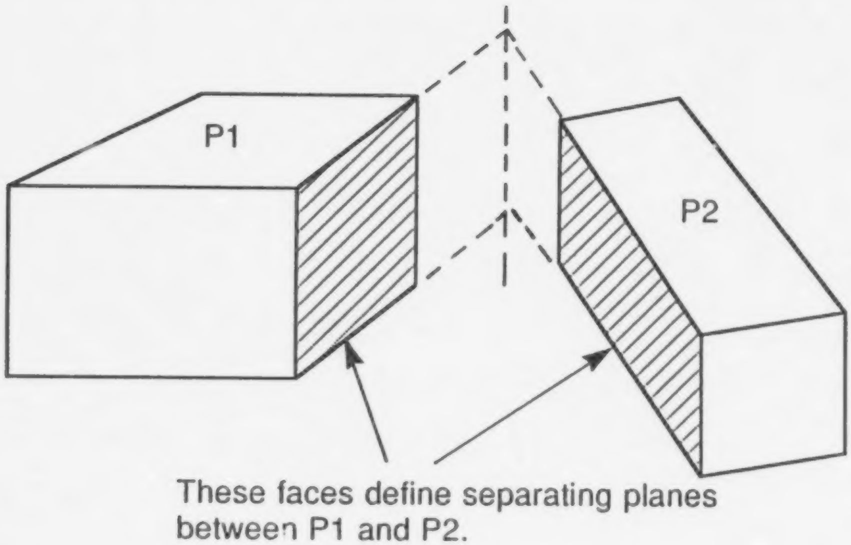

P1
P2
These faces define separating planes between P1 and P2.

Figure 6-6. Faces Defining Separating Planes.

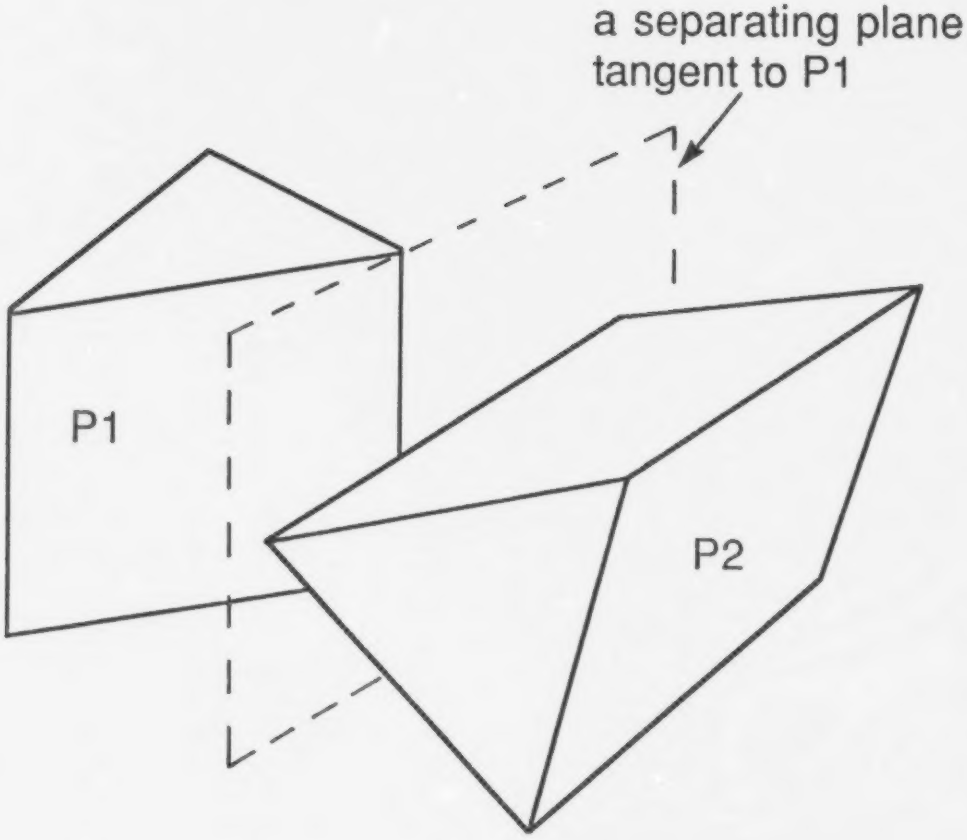

a separating plane
tangent to P1
P1
P2

Figure 6-7. No Face Defines a Separating Plane.

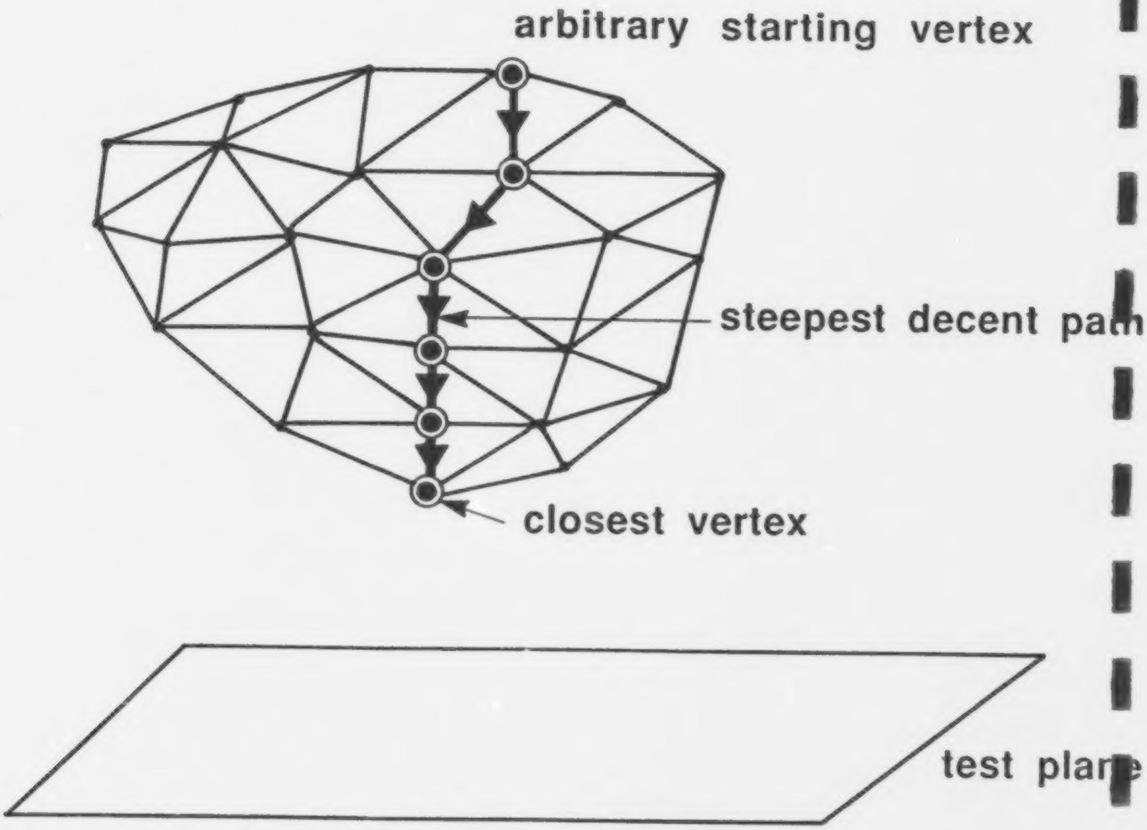

arbitrary starting vertex
steepest decent path
closest vertex
test plane

Figure 6-8. A Steepest Descent Search for the Closest Vertex.

minimum distance from P to a vertex of POLY2. The sign of the distance is determined by the orientation of P.

The closest vertex can also be found by the steepest descent method. Each node is now a vertex, and the branch nodes are adjacent vertices. The distance from each adjacent vertex to the plane is calculated, and the vertex with the minimum distance becomes the new current vertex. This procedure is repeated until the current vertex is closer than any adjacent vertices. This distance is the global minimum, and this vertex is the one closest to the plane. A typical search path is illustrated in Fig. 6-8.

6.3.5 Steepest Descent Algorithm

```
Open a starting node Ncurr and evaluate Vcurr.

UNTIL there are no new adjacent nodes or Ncurr is the best DO
    Open all nodes adjacent to Ncurr not in the previous list, and
    evaluate them.
    Add the new nodes to the previous list.
    Save the best node, Nbest, and its value, Vbest.

    IF Vbest is better than Vcurr THEN

        Ncurr=Nbest
        Vcurr=Vbest

    ELSE

        Ncurr is the best.

    ENDIF
END UNTIL

Ncurr is the best node, and Vcurr is its value.
```

6.3.6 Finding the Optimum Plane, and the Shortest Separating Distance.

At the conclusion of the best face search, we have identified the best face of POLY1 and the closest vertex of POLY2. The shortest connecting vector connects the best face to an edge containing the closest vertex. There are a limited number of cases which must be evaluated. These cases are illustrated in Fig. 6-9. The case which results in the shortest distance is the desired result. The optimum plane P_{opt} is normal to the shortest connecting vector. We can now evaluate DMIN(P_{opt}). If the result is negative, then the polyhedra intersect as illustrated by Fig. 6-10.

Algorithm:

```
IF   Vclos belongs to a face Fclos of POLY2 which is parallel to
     Fbest THEN

     IF   any vertex of Fclos projects onto Fbest, or any vertex of
          Fbest projects onto Fclos THEN

          The separating distance is the distance between the
          planes.

     ENDIF

ELSEIF Vclos Projects onto Fbest THEN

     The separating distance is the distance from Vclos to the plane
     of Fbest.

ELSE

     Find the separating distance between each edge of Fbest and each
     edge of POLY2 adjacent to Vclos.
     The minimum distance gives the shortest separating vector.

ENDIF
```

6.3.7 Calculation of Vertex to Face Distance

6.3.7.1 Projection of a Point onto a Plane. A point is projected onto a plane along a line normal to the plane and containing the point. The

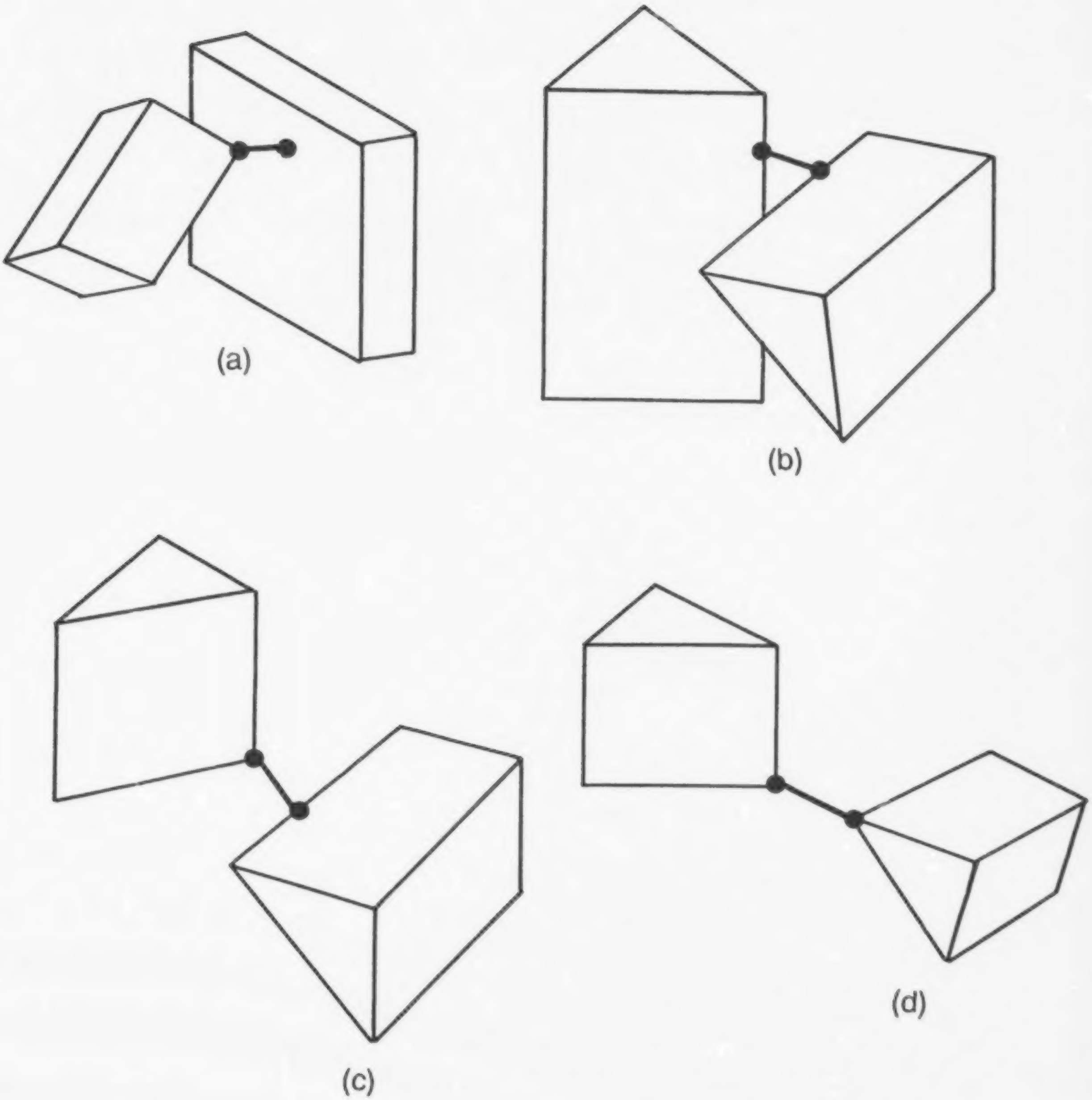


Figure 6-9. The Shortest Separating Vector Between Two Polyhedra.

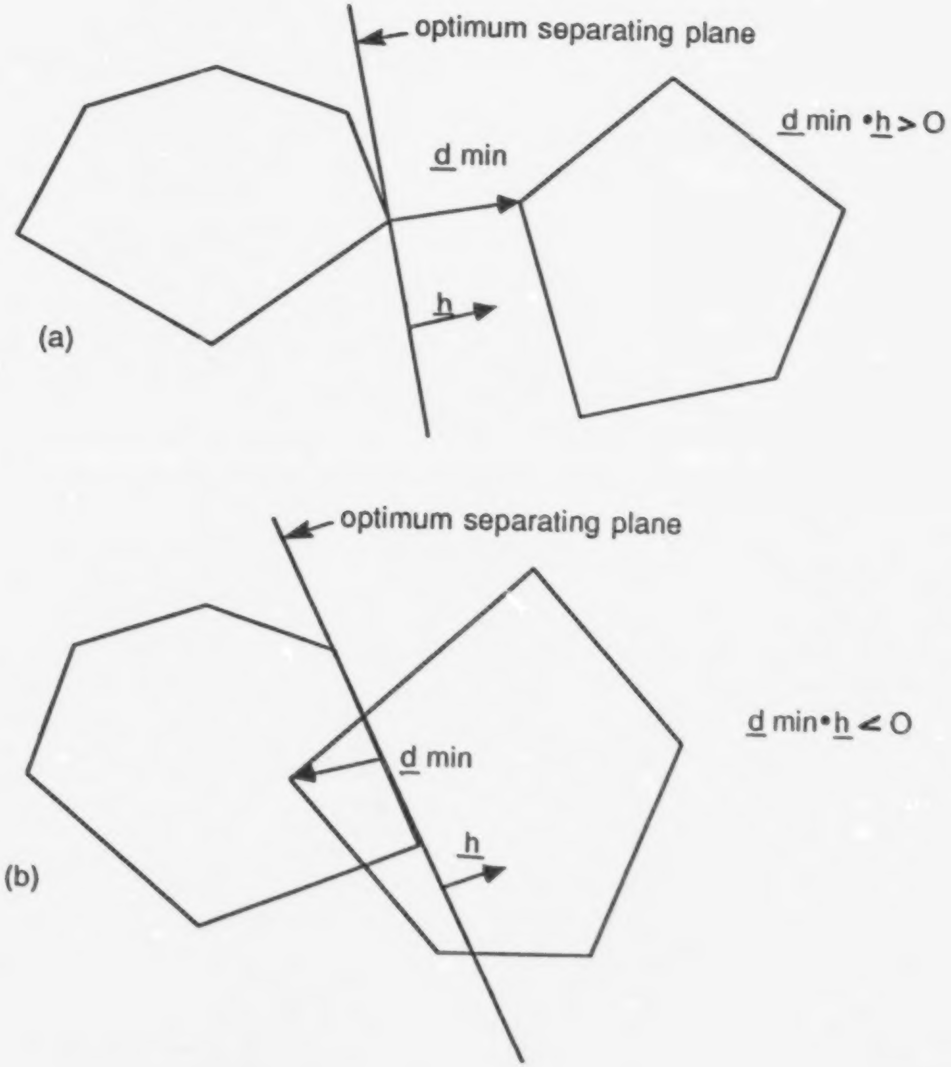

optimum separating plane
d min
d min • h > O
h
(a)
optimum separating plane
d min•h < O
d min
h
(b)

Figure 6-10. Separation and Interference Cases.

projection on the plane is the intersection of this line and the plane. The shortest connecting vector is the projection vector if the projection lies inside the boundaries of the face.

6.3.7.2 Face Inclusion Test. If a point P lies inside a convex polyhedral face F, then any other plane containing P and not containing F intersects the polygonal boundary of F at exactly two points. The two points define the ends of a line segment, which also contains P. This test is illustrated in Fig. 6.11.

Algorithm:

> Define a plane parallel to two of the reference coordinate axes and containing P.
>
> Starting at an arbitrary vertex of the face polygon, test each edge of the polygon for intersection with the plane.
>
> If the number of intersections is zero, P is outside the polygon.
>
> If the number of intersections is not zero or two, perturb the position of P slightly and try again. This handles the situation when a polygon vertex lies on the plane. In this case, the intersection may be counted twice, once for each edge containing the point.
>
> If the number of intersections is two, then define two coordinate axes parallel to the frame axes and define P as the origin. P lies inside the polygon if the two intersection points lie in opposite quadrants of this reference frame. This is equivalent to testing if P lies between the two intersection points.

6.3.8 Edge to Edge Separating Distance If the shortest separating distance is not the distance from the closest vertex to the best face plane, then it must be the shortest distance from an edge of the face to an edge containing the closest vertex. These edges are shown in Fig.

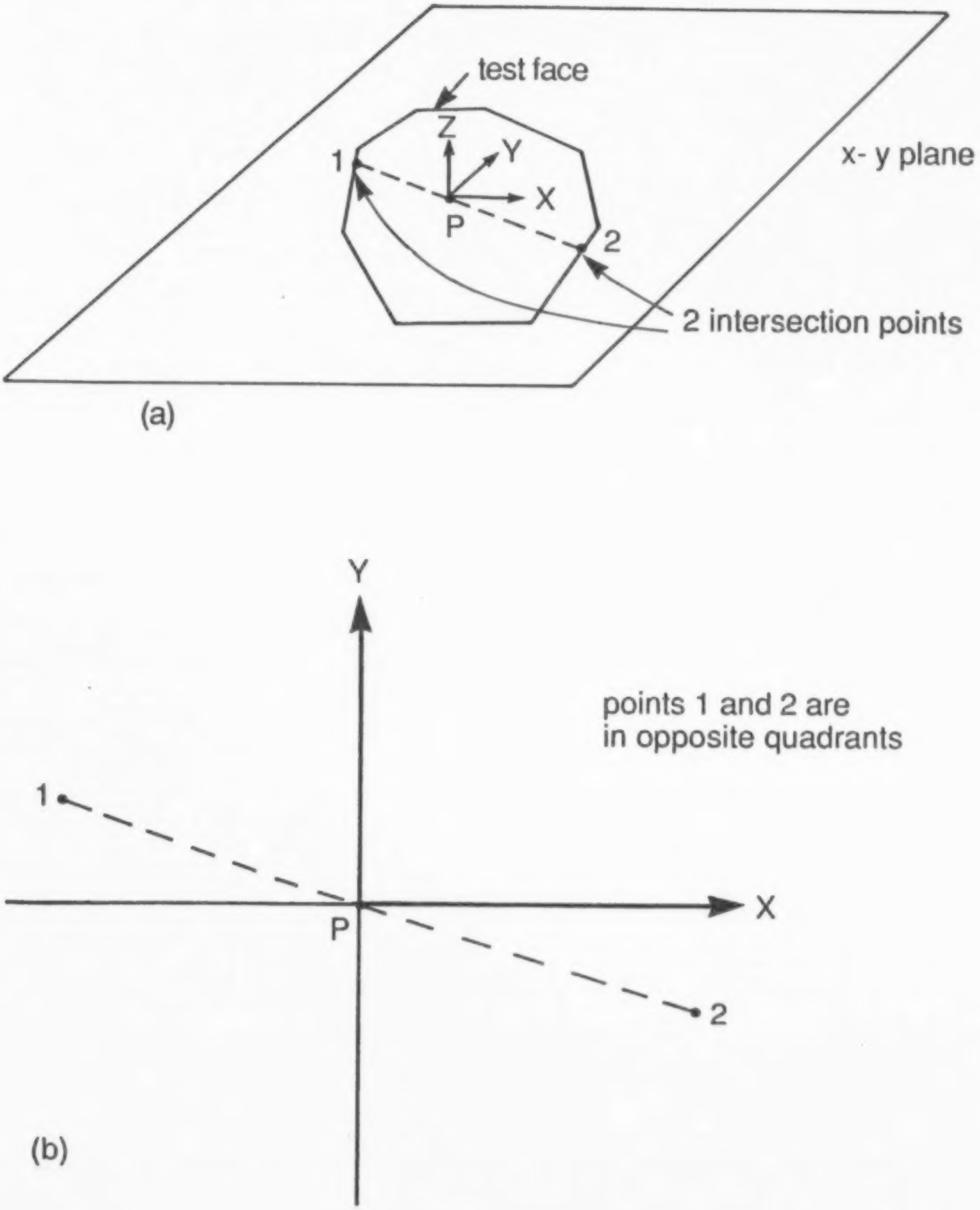

test face
Z
Y
1
X
x- y plane
P
2
2 intersection points
(a)
Y
points 1 and 2 are
in opposite quadrants
1
X
P
2
(b)

Figure 6-11. A Face Inclusion Test.

6-12. To find the minimum, each face edge is tested against each vertex edge using the following edge-edge algorithm.

The shortest vector connecting two lines in space is their common normal (Fig. 6-13). Given the two edges defined by endpoints p, q, let

$$\underline{U1} = \underline{q}1 - \underline{p}1 \quad \text{and} \quad \underline{U2} = \underline{q}2 - \underline{p}2.$$

We know that

$$\underline{p}1 + \lambda_1\, \underline{U1} = \underline{p}2 + \lambda_2 \underline{U2} + \underline{d}. \qquad (1)$$

We also know that the common normal $\underline{d}$ is parallel to $\underline{U1} \times \underline{U2}$,

$$\underline{d} = k\,(\underline{U1} \times \underline{U2}), \text{ where k is some constant} \qquad (2)$$

We can show that

$$|\underline{d}| = \left| \frac{(\underline{p}2 - \underline{p}1) \cdot (\underline{U1} \times \underline{U2})}{|\underline{U1} \times \underline{U2}|} \right| \qquad (3)$$

This gives the magnitude of $\underline{d}$, but the sign is not known.

A singularity occurs if the two lines are parallel. In that case, $\underline{U1} \times \underline{U2} = 0$, and another method must be used.

We can rearrange 1,

$$\lambda_1\, \underline{U1} - \lambda_2\, \underline{U2} = (\underline{p}2 - \underline{p}1) + \underline{d}. \qquad (4)$$

If we assume a sign for $\underline{d}$, we get a set of 3 equations in 2 unknowns. To solve, we choose two components i, j such that

$$\begin{vmatrix} (U1)i & (U2)i \\ (U1)j & (U2)j \end{vmatrix} \neq 0.$$

We can solve the two equations in two unknowns using Cramers rule. The values of λ_1, λ_2 are then inserted into the remaining equation. If the two sides are not equal, the assumed sign of $\underline{d}$ is wrong. In that case, the sign of $\underline{d}$ is then reversed and the above calculation repeated.

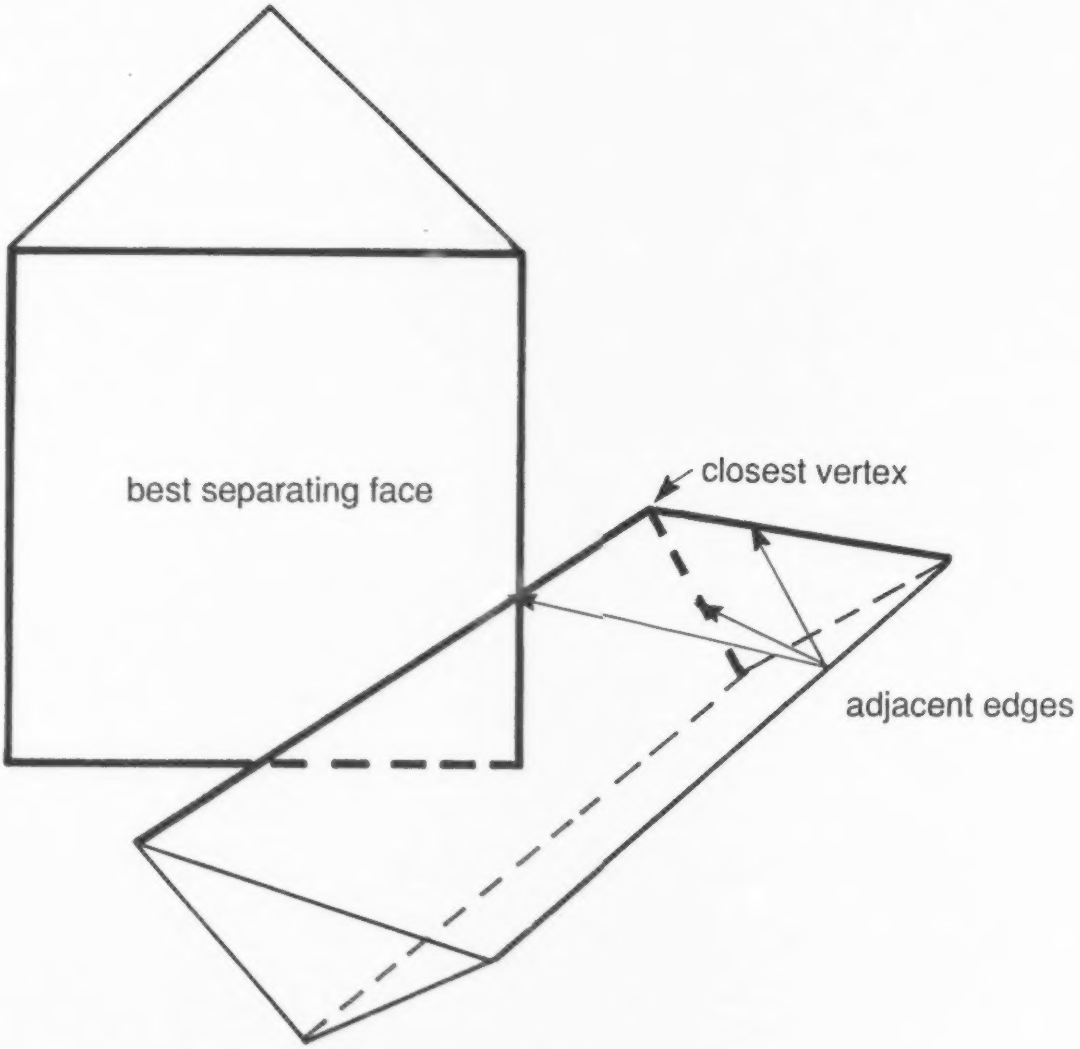

best separating face
closest vertex
adjacent edges

Figure 6-12. Face Edges and Vertex Edges.

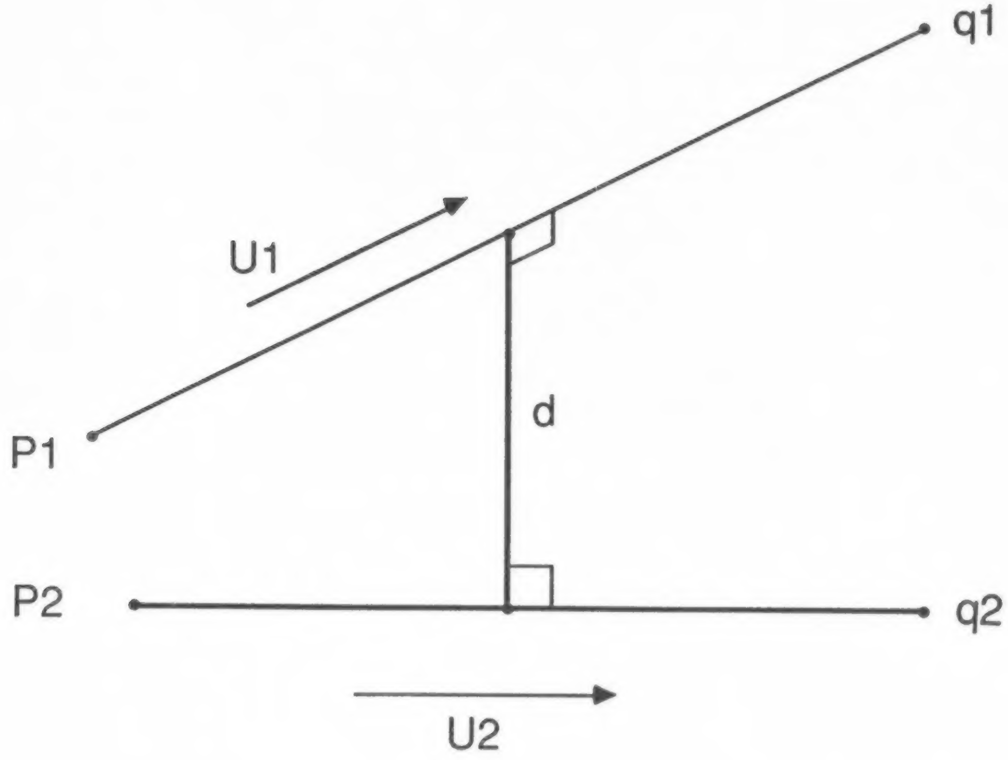

q1
U1
d
P1
P2
q2
U2

Figure 6-13. The Common Normal of Two Lines.

There are four possible cases:

1. IF $(0 < \lambda_1 < 1)$ .and. $(0 < \lambda_2 < 1)$ THEN

   The common normal intersects both edges, and is the shortest connecting vector.

2. Else

   The shortest connecting vector STARTS at an endpoint of the FIRST edge and is normal to the SECOND edge.

   Project each endpoint onto the line of the opposite edge. If the projection lies on the edge, it represents a connecting vector.

3. Else

   The shortest connecting vector ENDS at an endpoint of the SECOND edge and is normal to the FIRST edge.

   Same as above, with the edges reversed.

4. Else

   The shortest connecting vector joins an endpoint of the first edge to an endpoint of the second edge.

   Connect all pairs of endpoints and select the shortest resulting vector.

These edge-edge cases are shown in Fig. 6-14.

6.3.9 Projection of a point onto a line. Suppose we have a line segment with endpoints $\underline{E1}$, $\underline{E2}$ and a point $\underline{P}$ not on the line. $\underline{P}'$ is the projection of $\underline{P}$ onto the line.

Let

$$\underline{U} = \underline{E2} - \underline{E1}$$

and

$$\underline{V} = \underline{P} - \underline{E1}.$$

Let

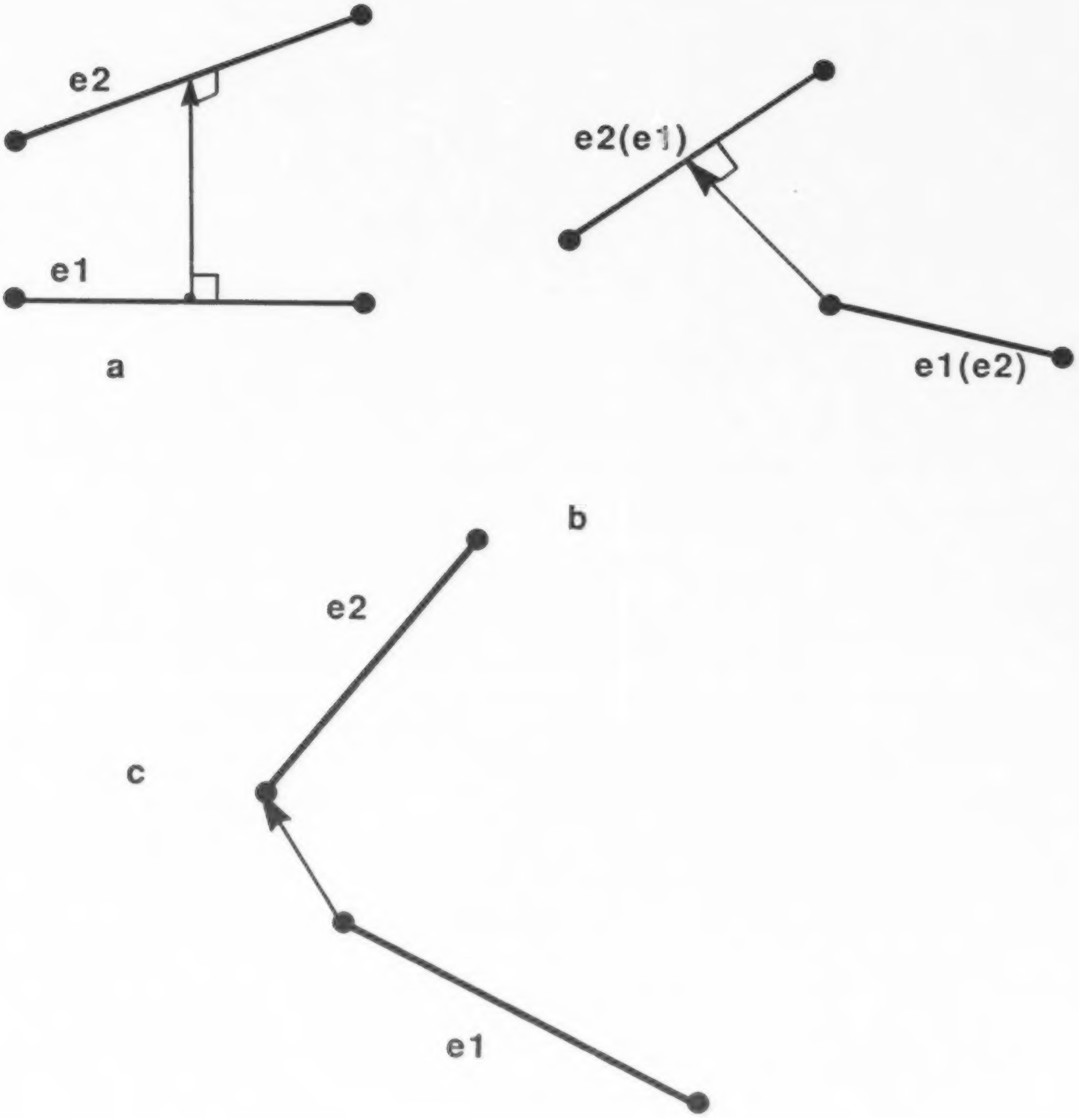

e2
e1
a
e2(e1)
e1(e2)
b
e2
c
e1

Figure 6-14. Shortest Connecting Vectors for Two Edges

$$\lambda = \frac{|\underline{P}' - \underline{E1}|}{|\underline{E2} - \underline{E1}|}$$

We can show that

$$\lambda = \frac{\underline{U} \cdot \underline{V}}{\underline{U} \cdot \underline{U}} .$$

If $0 < \lambda < 1$ then $\underline{P}'$ lies on the line segment.

6.3.10 Possible Improvements. The execution time of the separating plane algorithm is expected to increase exponentially with the number of parts to be tested. This is an important limitation since it may be necessary to represent simple concave shapes by clusters of many convex parts. The algorithm can be improved by adding another level of hierarchy. If the convex hulls of the clusters are calculated and stored, then they can be tested as convex parts. Only the convex hulls which intersect need to be decomposed into their parts and tested further. This method will not provide hidden surface removal, but interference detection will be accelerated greatly in most cases.

6.4 Comparison and Assessment of Interference Detection Methods

The important features of each interference detection method can be summarized at this point.

1. Face Intersection Method

   This method has the following features:

   a. A hierarchy of tests are used.

   b. Interference can be detected for concave polyhedra.

   c. The intersection volume can be found for convex polyhedra. This intersection can be shown graphically.

   d. Hidden surface removal is also performed.

The important disadvantages are:

a. The method is slow since it works on faces rather than polyhedra.

b. The execution time is expected to increase exponentially with the number of faces.

2. Duncan Method I

This method has the following features:

a. The algorithm is simple and general.

b. The interference volume can be shown graphically as a 'cloud' of points on the surface of one of the solids.

c. The densely defined polyhedron can be concave.

d. The method can be simplified to a true/false interference test.

e. The execution time increases linearly with the number of faces of the sparse polyhedron.

The disadvantages of this method are:

a. A fine mesh of possibly redundant points must be stored for one of the objects. This is inefficient for most regular engineering shapes which can be sparsely defined.

b. The number of required calculations is not greatly reduced for cases where the polyhedra are simple.

c. The method uses exhaustive testing of points, which is inefficient.

d. The reliability of the method depends on the spacing of the surface points. The possibility of undetected interference exists.

e. The sparsely defined polyhedra must be convex or one of a restricted set of concave polyhedra.

f. Objects are separated into two sets, moving and stationary polyhedra. This is an artificial restriction which assumes that objects belonging to the same set cannot intersect.

3. Duncan Method II

This method has the following features:

a. Concave polyhedra can be handled.

b. Interference is reliably found.

c. No redundant data is needed.

The method has the following disadvantages:

a. The case of total inclusion of a polyhedron by another is not considered. This can be added as a simple extension, and is not generally an important case.

b. The intersection volume is not found.

c. The algorithm execution time is expected to increase exponentially as the number of polyhedral faces increase.

d. Exhaustive evaluation of all possible edge-face combinations must be performed.

Both of the Duncan methods make use of a geometric representation based on a Mercator mapping. This mapping is not appropriate for many polyhedra and leads to the addition of redundant points for consistency. This representation is not consistent with the geometric representation chosen for this work. Both methods use a preliminary bounding box test for interference.

4. Separating Plane Method

This method has the following features:

a. The algorithm is hierarchical to optimize speed and efficiency.

b. Fast searching is performed by a steepest descent approach. Exhaustive search is not needed. This drastically reduces the number of required calculations.

c. The execution speed is expected to increase linearly with the number of polyhedral faces or vertices.

d. This method can perform either a true/false interference test, or it can calculate the shortest connecting vector between two disjoint polyhedra.

This method has the following disadvantages:

a. Only convex polyhedra are allowed.

b. The intersection volume is not found.

c. Calculation of the shortest connecting vector is much slower than a simple interference check.

All of these methods have some features in common:

a. They work on polyhedral representations of real solids.

b. These methods consider discrete positions of objects. Motion is approximated by a series of discrete positions.

c. All of the algorithms are expected to have execution times which grow exponentially with the number of parts to be compared.

Based on the preceeding assessment, the separating plane method has been chosen as the best method. Calculation of the intersection volume is not necessary except for a graphical depiction and the capability of calculating shortest separating vectors is an important tool for further work in optimization to avoid collisions. The method is efficient for determining if two polyhedra intersect, and a hierarchical processing strategy can be devised to improve the performance of the algorithm when many polyhedra must be compared.

7. PATH PLANNING

7.1 Iterative Search for a Feasible Welding Path

This method was developed under a previous contract [Buchal et al 1984]. It has been extended by including interference as an additional constraint.

The user specifies the workpiece position and the seam to be welded. This defines a path in space for the welding torch. However, there are several robot configurations for a given torch position. Also, the torch is rotationally symmetric, which adds a degree of freedom to the solution. A feasible solution is found by the following trial and error strategy.

For a chosen solution configuration and an arbitrary torch rotation, an inverse solution is found for the first weld point. If it is kinematically feasible and does not interfere, a solution with the same configuration is found for the next point. This is repeated until a joint limit is exceeded or interference occurs. The torch is then rotated in increasing arcs in both directions from its starting orientation until a feasible, non-interfering solution is found, or until a rotation limit is exceeded. The rotation step size and maximum limits can be specified in the setup files. If no feasible solution is found, the robot returns to the start of the weld and the process is repeated until feasible solutions are found for the entire path, or until all configurations have been tried.

This algorithm can be stated more concisely as follows:

```
Start at first configuration.
UNTIL no more configurations or success DO

     Start at first weld point.
```

```
      UNTIL end of weld or fail DO

           Find inverse solution.
           Test joint limits and interference.

           IF fail THEN

           UNTIL limit or success DO

                     Rotate torch about it axis.
                     Find inverse solution.
                     Test limits and interference.

                END UNTIL

           END IF

           IF success THEN

                Advance to next weld point.

           ELSE

                Try next configuration.

           END IF

      END UNTIL

END UNTIL

IF fail THEN

     Reposition the workpiece and try again.

END IF
```

7.1.1 Extension to a Moving Workpiece. The previous method can be used without modification where a known motion of the workpiece occurs during the weld. The instantaneous position of the workpiece must be calculated for each weld position in this case.

A moving workpiece is specified by the user by defining an initial and final table position for each weld. The motion is uniform and continuous between these endpoints. Thus, the workpiece position is a function of the distance along the weld.

7.2 Inter-weld path

A complete welding program consists of welding trajectories, separated by trajectories from the end of one weld to the beginning of the next weld. These inter-weld paths must also be feasible and interference free. Unlike the actual welding path, only the initial and final positions are fixed. Thus, there are an unlimited number of solutions.

The simplest trajectory is a joint-interpolated trajectory where all joints are moved simultaneously from their initial to their final positions. The joint velocities are constant, and are set so that all the joints reach their final positions at the same time. To achieve minimum elapsed time, the joint velocities are determined by the smallest joint velocity limit.

A joint-interpolated path joining the last position of the previous weld to the first position of the next weld is a minimum time path. We will call this the fundamental path. The fundamental path may be unpredictable and convoluted, expecially for large joint angle changes. Kinematic feasibility is always assured, but interference may occur.

The inter-weld path can be further constrained by adding additional positions between the initial and final positions. This is equivalent to defining a point-to-point joint-interpolated path with more points. The path joining two adjacent positions is called a segment.

To interactively generate an interference free path, the joint variables are interpolated along each segment, and interference checking is performed. If interference is detected along a path segment, then an extra position can be inserted between the two endpoints. This replaces the current segment with two new segments, which must both be checked. If no interference is detected, then no further attention is required by that segment. Positions can be added, moved or removed until an acceptable inter-weld path is obtained. In practice, new positions can be easily generated by moving the robot joint by joint away from an interpolated position which interferes. The resulting path should follow the fundamental path as closely as possible in order to minimize the elapsed time. The elapsed time of the fundamental path is the lower bound. Fig. 7.1 illustrates this idea.

7.3 Path Planning for Kinematic Systems with Redundancy

This chapter addresses some advanced topics which have not yet been considered in detail. The problems discussed are central to the development of a more powerful path planning method.

Let us define a general kinematic system with m independent task coordinates, x(m), and n independent joint coordinates, q(n). In general, no explicit inverse solution of the form q=f(x) exists.

A linearized equation can be formulated for incremental changes in the coordinates:

$\delta\underline{x} = J\delta\underline{q}$, where J(q) is the Jacobian matrix.

Define an inverse G such that

$$\delta\underline{q} = G\delta\underline{x}$$

The existence of G depends on the dimensions m and n.

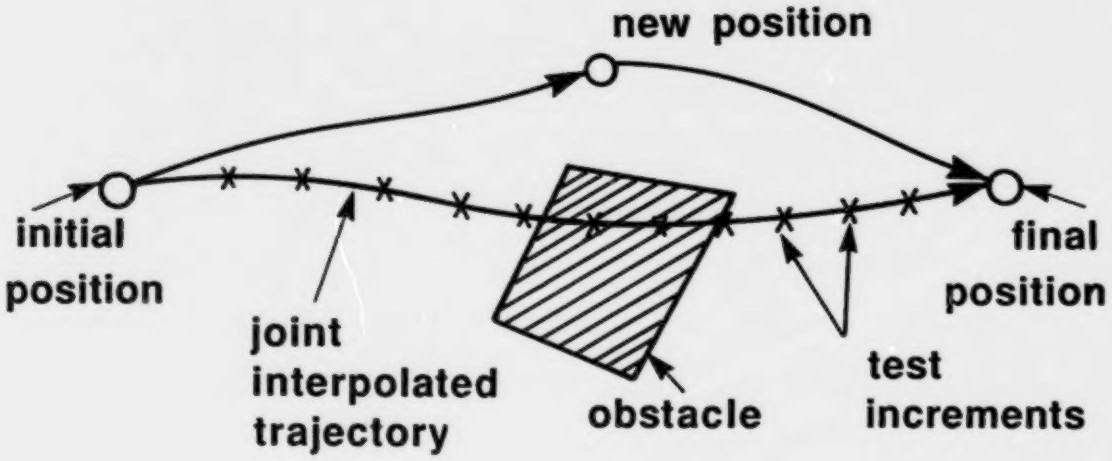

new position
initial position
joint interpolated trajectory
obstacle
test increments
final position

Figure 7-1. Generation of an Inter-weld Path.

If $m < n$, the system is overconstrained and in general no solutions exist.
If $m = n$, the system has a finite number of unique solutions. In this case, $G = J^{-1}$ and solutions exist if J^{-1} exists.
If $m > n$, there are an infinite number of solutions and the system is said to be redundant.

A partial survey of the literature in this area incudes [Whitney 1969], [Whitney 1972], [Fournier and Khalil 1977], [Konstaninov, Markov and Nechev 1981], [Konstantinov, Genova and Zahariev 1981], [Konstantinov et al 1982], [Vukobratovic and Kircanski 1982], [Kircanski and Vukobratovic 1984].

7.3.1 Redundancy and Local Optimization The existence of redundancy allows solutions to be chosen which optimize some criteria. Using classical optimization, the joint velocites or the kinematic energy can be minimized. A quadratic performance index can be formulated as

$$\phi = 1/2\ \delta\underline{q}^T M \delta\underline{q}$$

where M is a weighting matrix.
The constraints can be stated as

$$\delta\underline{x} - J\delta\underline{q} = 0.$$

The optimal solution is

$$\delta\underline{q} = M^{-1}J^T(JM^{-1}J^T)^{-1}\delta\underline{x}.$$

If $M = I$, this reduces to the minimum norm solution

$$\delta\underline{q} = J^T(JJ^T)^{-1}\delta\underline{x}.$$

where $J^T(JJ^T)^{-1} = J^+$, the pseudo-inverse.

A technique for maximizing joint availability is presented by [Konstantinov, Markov, Nechev 1981]. This method imposes a cost for approaching joint limits.

A method for avoiding interference has been proposed by [Kircanski, Vukobratovic 1984]. In this method, the exact separating distances between the robot links and the obstacles are calculated. If a link approaches closer than some threshold distance, then several strategies are proposed:

1. Motion of the joints which affect the link in question are penalized as the link approaches the obstacle. The result is that the link is stopped before it hits the obstacle. The task must then be performed by the remaining free links.
2. Motion of the closest point on the link is constrained. Again, sufficient degrees of freedom must remain to complete the task.
3. Finally, a performance index is proposed which would attempt to move the closest point away from the obstacle.

No published work has appeared which considers optimization to avoid both joint limits and interference, but it is believed that the techniques described above can be combined in a straightforward way.

7.3.2. Global Path Planning. The optimization methods discussed above apply locally only, and do not insure that the final path is optimum. The path resulting from following the locally optimum trajectory depends on the starting configuration. In other words, there are an infinite number of paths which are locally optimum. Only a subset of the possible starting configurations may yield a feasible path. Also, a globally optimum path may be locally sub-optimal. An important question is how to find an optimum, or even an acceptable path subject to global constraints and performance criteria. Task planning and search techniques from the field of artificial intelligence may be applicable to this problem.

8. GRAPHICS METHODS

8.1 Hidden Surface Removal Algorithm

To obtain a solid, shaded graphical depiction of a scene, hidden surface removal is necessary. For a scene composed of convex polyhedral parts, it is sufficient to order the parts from back to front for output to the screen. A convex polyhedron cannot obscure itself, so the visible faces of a given part can be drawn in any order. This algorithm is a simplified adaptation of the Newell/Newell/Sancha hidden surface algorithm [Rogers 1985] for polygons. The following algorithm is used to sort the parts for output:

```
Sort the unordered parts list by maximum screen depth z.
Define part P as the part at the head of the unordered list.
UNTIL all parts are ordered DO
    Find the set of unordered parts which overlap part P in depth
    z.
    Eliminate those whose bounding boxes are disjoint in x, y.
    Eliminate those parts which are on the near side of a separating
    plane between the test part and part P.
    Eliminate those parts which do not overlap part P in screen
    space.
    IF all the parts have been eliminated, THEN
        Add part P to the output list and remove it from the
        unordered list. The next part in the list becomes part P.
    ELSE
        Replace part P at the head of the list by the first part
        which is obscured by part P.
        Insert part P into the list according to maximum depth.
    ENDIF
END UNTIL
```

This algorithm generates a list of parts ordered by screen depth.

The following algorithm is used for output to the screen:

```
UNTIL all parts output DO
      Get the next part from the ordered list.
      FOR each face of the part DO
            Calculate the dot product of the line of sight and the
            face normal.
            IF the face is visible THEN
                  Assign a shade of gray according to the angle of
                  the face from the line of sight.
                  Output the face as a filled polygon.
            ENDIF
      ENDFOR
END UNTIL
```

8.2 Optimization for Speed

Fast program response is necessary for an interactive system. Response time is a function of the speed of hardware, the efficiency of the algorithms, and the amount of processing required. The interactive program runs in a loop, with a new frame generated during each pass. Speed has been maximized by eliminating all unnecessary processing from inside the loop, depending on the current situation. For example, for a static display, no new graphical processing is necessary. Similarly, much less processing is required to generate a wire frame image than to generate a shaded solid image.

8.2.1 Fortran vs. C. Several simple benchmark programs were written in both Fortran and C to compare their running speed. In every case, Fortran ran the same or faster than C. Simple operations were performed repetitively in both Fortran and C. See Appendix V.4 for the source listings. The comparative execution times are shown in Table 8.1. It is clear that in general Fortran runs much faster on the IRIS.

TABLE 8-1. Benchmark Comparisons Between Fortran and C.

Operation	Time (sec)	
	Fortran	C
floating point operations	24	22
integer operations	11	10
data moves	12	26
nested loops	4	12
single depth loop	4.5	11.5

8.2.2 Hardware. AUTOWELD makes use of the IRIS hardware matrix multiplication wherever possible. The hardware can be upgraded to further improve the execution speed. A floating point processor will increase the speed of floating point operations, and the latest processor board will increase the program execution speed. Depending on the program, up to an order of magnitude improvement can be expected.

There are no repetitive operations in AUTOWELD that would justify a custom hardware implementation.

8.2.3. Efficiency of Algorithms. Many simple algorithms which solve a problem in an obvious way are very inefficient. This can lead to execution times which increase exponentially with the complexity of the problem. Great improvements in speed can be realized by adopting more sophisticated algorithms. For example, for interference detection the number of separating planes which must be considered increases with the square of the number of parts. Thus, the program will run much slower with only a moderate increase of complexity of the scene. Precalculating known separating planes will help only slightly. A more effective approach is to use heuristics and other knowledge to prune the number of calcalulations required by eliminating redundant cases.

The costs of these kind of improvements are increased development effort, and more complicated algorithms and data structures.

9. DESCRIPTION AND ASSESSMENT OF SOFTWARE PACKAGE 'AUTOWELD'

9.1 Design Approach

AUTOWELD is designed with the aim of utilizing the capabilities of the Silicon Graphics IRIS computer. Speed and efficiency are the prime goals. Transporting AUTOWELD to another graphics machine requires replacing all of the graphics calls with the appropriate analogues, or writing a library of interface routines.

AUTOWELD was written in Fortran 77. A modular, structured approach was taken. Each subroutine is well defined, with clear inputs and outputs.

9.2 Features and Capabilities

Given a CAD specification of the workpiece geometry and seams, AUTOWELD interactively generates a complete welding program composed of weld trajectories connected by joint-interpolated inter-weld trajectories. The program consists of a record of robot and positioner joint values and velocities for closely spaced points. The user controls the following parameters:

- definition of desired weld path
- positioning of workpiece
- specification of welding sequence
- specification of inter-weld paths

AUTOWELD provides the following tools:

- automatic generation of a feasible welding trajectory for a desired weld
- interference detection

- user-friendly graphics environment for fast interaction and clear understanding
- graphical simulation of the program for verification.

The following features are incorporated by AUTOWELD:

1. Definition of desired weld paths

The user can interacively define the seams which are to be welded, including direction, by picking points on the CAD reference seams.

2. Weld generation

The user can interactively position the workpiece, and pick a seam to be welded. AUTOWELD will automatically search for a feasible welding trajectory, which will be stored in the welding program.

The user can specify a synchronously moving workpiece by specifying the starting and ending table position for a given seam. The workpiece will then move smoothly and continuously while the weld progresses.

The user can perform the following editing functions:

- unweld a seam
- unweld all welded seams
- delete a seam definition
- delete all seam definitions
- reverse the direction of a seam definition.

3. Inter-weld paths

The last position of the previous weld trajectory and the first position of the next weld trajectory define two positions which specify the endpoints of a joint-interpolated move between the welds.

The user can select an inter-weld path for verification or modification by picking the weld preceding the path. In an optional mode, the user can pick the weld following the desired inter-weld path. This is useful for selecting the path from the initial position to the start of the first weld.

The user can step through the defined inter-weld positions.

A joint-interpolated move between two adjacent positions can be simulated. The user can control the step size of the interpolation.

The user can interactively modify a particular inter-weld path by:

a. Inserting a new position in the path sequence. A new position can be generated by moving the robot or table joint by joint.

b. Removing a position from the path.

c. Relocating a position on the path.

4. Graphics

- Wireframe or shaded solid image can be selected at any time.
- The CAD seam can be highlighted for better visibility.
- The order of completed welds can be shown.
- The view point can be changed interactively.

5. Verification

- The welding program can be simulated graphically at any stage.

- Elapsed welding and total time are shown as the program is replayed.
- Welding parameters are shown during replay of weld trajectories.
- If desired, AUTOWELD will test for interference during replay, and will store those inter-weld paths which interfere. Those paths can then be modified interactively by the user.
- The user can skip over the current inter-weld path or weld trajectory at any point during replay. This allows the user to quickly scan forward to the path of interest.
- All graphics controls are enabled during replay.
- The user can step through any selected inter-weld path to test for interference.
- Exact separating distances can be calculated and shown.
- Interference is indicated by a change in colour of the wire-frame image, or by the solid image becoming wire-frame. Interfering pairs of parts are shown in a contrasting colour for clarity.

6. User interaction

User interaction is via the mouse, and buttons on the keyboard. A menu display on the screen indicates the available options. These input commands are processed by a single subroutine, which sets appropriate mode flags. The mode of the program is determined by the values of the mode flags.

9.3 Performance Assessment for Several Test Cases

A robot kinematically similar to a Unimation PUMA, and a generic positioning table with three rotational degrees of freedom were specified. This workstation definition is specified by editing data files. A trial CAD seam definition was generated explicitly, and several workpiece configurations were modelled (Fig. 9.1).

The robot was represented by three clusters of four parts total. The positioning table was represented by a single part. The test workpieces ranged in complexity from two parts to ten parts. There does not appear to be any significaant difference in the interactive performance of AUTOWELD for the different cases.

9.4 Auxiliary Software

Two auxiliary programs have been written. These are described below.

9.4.1 GEM- A Solid Modelling Program.

A simple solid modelling package has been written to facilitate the generation of geometric models for several different test workpieces. The program is very limited, but it is much faster and less error prone than specifying geometric data manually using the editor. GEM is described in detail in Appendix IV.

9.4.2 GENPRO- A post-Processor to Generate a Pseudo-program for Welding

The output of AUTOWELD resides in files which are not readable by the user. These files contain all information required for the generation of a welding program for a workstation. GENPRO is a simple program which reads the output of AUTOWELD and converts it into a pseudo-program in

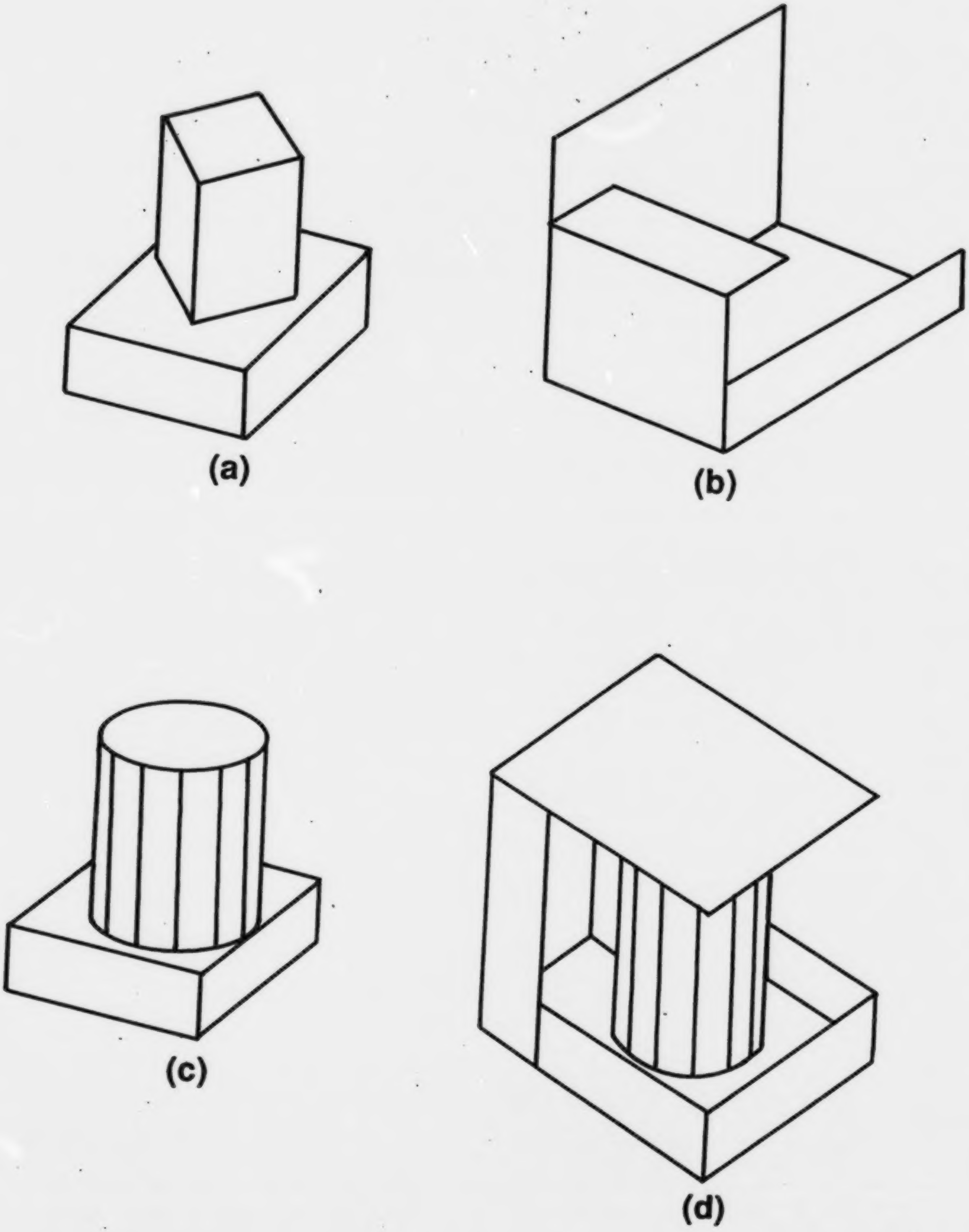


Figure 9-1. Sketches of the Test Workpieces.

tabular form. The output of GENPRO is stored in file PROGRA.DAT. The following is a sample listing of PROGRA.DAT for a typical program.

```
ROBOT WELDING PROGRAM GENERATED OFFLINE BY AUTOP


************** WELD 1 *********

*****Interweld positions *******

           Robot joint positions                         Table joint       Time
                                                          positions      intervals

 1       .0      .0      .0      .0       .0      .0     16.4    9.5     .0    .00
 2    131.8 -94.3-171.7   62.46    83.3-137.8           43.5    1.0     .0   1.78
 3    151.5                                              44.4     .7     .0    .24

***** Weld trajectory *****

          Robot joint positions                          Table joint       Time
                                                          positions      intervals

 1    151.5 -97.3-172.01 64.6     86.0-142.3            44.4     .7     .0    .00
 2    150.4 -96.9-172.1  65.1     88.4-144.1            42.8    1.2     .0   1.75
 3    149.3 -96.4-172.2  65.5     91.0-145.9            41.1    1.7     .0   1.75
 4    148.3 -95.9-172.4  65.8     93.4-147.7            39.5    2.3     .0   1.75
 5    147.4 -95.3-172.6  65.9     95.8-149.6            37.8    2.8     .0   1.75
 6    146.6 -94.7-172.8  66.0     98.0-151.4            36.2    3.3     .0   1.75
 7    144.7 -95.7-169.7  71.8     97.3-147.4            34.5    3.8     .0   1.75
 8    144.0 -94.9-170.0  71.7     99.5-149.1            32.9    4.3     .0   1.75
 9    142.5 -95.8-167.1  77.0     98.6-144.5            31.2    4.8     .0   1.75
10    141.2 -96.7-164.3  82.0     97.1-139.7            29.6    5.4     .0   1.75
11    140.6 -95.7-164.8  81.7     99.5-141.1            27.9    5.9     .0   1.75
12    139.6 -96.6-162.1  86.3     97.6-135.9            26.3    6.4     .0   1.75
13    138.8 -97.6-159.6  90.6     95.3-130.7            24.6    6.9     .0   1.75
14    138.3 -96.5-160.2  90.2     97.7-131.8            23.0    7.4     .0   1.75
15    137.7 -97.5-158.0  94.0     94.8-126.1            21.3    7.9     .0   1.75
16    137.2 -96.4-158.7  93.6     97.2-127.1            19.7    8.5     .0   1.75
17    136.8 -95.2-159.5  93.0     99.6-128.1            18.0    9.0     .0   1.75
18    136.4 -95.9-158.5  93.7     98.6-128.8            16.4    9.5     .0   1.75
```

```
***** Welding parameters *****

voltage, wire feed rate, gas flow rate, current

   1      20.97    265.00     22.50   145.73
   2      20.97    265.00     22.50   145.67
   3      20.97    265.00     22.50   145.61
   4      20.97    265.00     22.50   145.55
   5      20.97    265.00     22.50   145.49
   6      20.97    265.00     22.50   145.43
   7      20.97    265.00     22.50   145.36
   8      20.97    265.00     22.50   145.30
   9      20.97    265.00     22.50   145.25
  10      20.97    265.00     22.50   145.21
  11      20.97    265.00     22.50   145.17
  12      20.97    265.00     22.50   145.13
  13      20.97    265.00     22.50   145.09
  14      20.97    265.00     22.50   145.07
  15      20.97    265.00     22.50   145.06
  16      20.97    265.00     22.50   145.05
  17      20.97    265.00     22.50   145.05
  18      20.97    265.00     22.50   145.00


********** WELD 2 **********

****  Inter-weld positions *****

            Robot joint positions                        Table joint      Time
                                                          positions     intervals

   1    2.6-104.8   28.2   59.7  -94.1   64.8          -10.5 -17.8    .0    .00
   2    1.3-106.5   29.9   61.6  -97.8   62.5          -10.5 -17.8    .0   1.75
   3   - .4-106.3   27.6   66.4  -95.1   66.0          -10.5 -17.8    .0   1.75
   4   -1.6-108.3   29.7   68.2  -96.4   68.2          -10.5 -17.8    .0   1.75
   5   -3.1-108.2   27.7   72.5  -96.4   68.2          -10.5 -17.8    .0   1.75
   6   -4.4-108.2   25.9   76.4  -93.8   72.5          -10.5 -17.8    .0   1.75
   7   -5.2-110.6   28.4   78.0  -97.8   71.7          -10.5 -17.8    .0   1.75
   8   -6.3-110.6   26.9   81.3  -95.2   76.5          -10.5 -17.8    .0   1.75
   9   -7.0-113.2   29.7   82.8  -99.1   76.2          -10.5 -17.8    .0   1.75
  10   -7.8-113.2   28.4   85.4  -96.4   81.5          -10.5 -17.8    .0   1.75
  11   -8.5-113.3   27.2   87.4  -93.4   87.2          -10.5 -17.8    .0   1.75
  12   -9.0-116.0   30.4   88.8  -97.2   87.5          -10.5 -17.8    .0   1.75
  13   -9.5-116.1   29.6   90.0  -94.1   93.6          -10.5 -17.8    .0   1.75
```

```
***** Welding parameters *****

voltage, wire feed rate, gas flow rate, current

   1      20.97   265.00   22.50  146.90
   2      20.97   265.00   22.50  146.78
   3      20.97   265.00   22.50  146.64
   4      20.97   265.00   22.50  146.50
   5      20.97   265.00   22.50  146.36
   6      20.97   265.00   22.50  146.22
   7      20.97   265.00   22.50  146.07
   8      20.97   265.00   22.50  145.92
   9      20.97   265.00   22.50  145.77
  10      20.97   265.00   22.50  145.62
  11      20.97   265.00   22.50  145.47
  12      20.97   265.00   22.50  145.32
  13      20.97   265.00   22.50  145.18


****** Return to rest ******

            Robot joint positions                   Table joint      Time
                                                     positions     intervals

   1  -9.5-116.1  29.6  90.0 -94.1  93.6            -10.5 -17.8    .0  1.75
   2    .0    .0    .0    .0    .0    .0             16.4   9.5    .0   .00

***************  END OF PROGRAM ****************
```

10. CONCLUSIONS

The main goals of this contract have been realized by a software package, AUTOWELD, which successfully incorporates solutions to many of the problems initially identified.

10.1 Analysis and Discussion of Work Completed

10.1.1 Geometric modelling. A flexible and general geometric modelling method has been specified. General solids are modelled by clusters of convex polyhedra. A fine or coarse approximation can be chosen, and simple polyhedra accurately model many engineering shapes. An external preprocessor is required to generate a consistent model of more complicted situations, to prevent the introduction of inconsistencies and errors into the representation. An external CAD system is required to generate the data for this model. An elementary preprocessor, GEM, has been written to simplify the generation of test models.

10.1.2 CAD seam specification. A compact and unambiguous method has been developed to specify a minimal CAD seam description.

10.1.3 Kinematics. A general method for specifying kinematic parameters and relationships has been defined. The method can be extended to any number of kinematic chains, each with any number of degrees of freedom.

The explicit inverse kinematics problem has been solved for a five or six degree of freedom robot belonging to a restricted class of industrial robot configurations. A strategy for finding a feasible path has been successfully applied. A general inverse kinemtic solution with optimization must be developed for situations with more than six degrees of freedom.

AUTOWELD places responsibility for positioning the workpiece on the user. If a feasible solution exists for welding a seam in a given position, it will be found by AUTOWELD. Otherwise, the workpiece must be repositioned by trial and error, based solely on the user's judgement.

A strategy for specifying and modifying inter-weld paths as joint interpolated motions has been developed and successfully implemented.

10.1.4 Interference detection. Four interference detection methods have been developed. The method chosen as best detects interference very quickly in most cases. The method is fastest when parts are well separated in space, and is slower as the parts come closer to each other. The method is fast enough for an interactive environment with a workstation of moderate complexity.

10.1.5 Expert welder. No further work has been done on the development of an expert welder. Parts of the previously developed expert welder [Buchal et al 1984] are incorporated into AUTOWELD for demonstration purposes.

10.1.6 Hidden surface removal. A simple hidden surface removal algorithm has been developed and implemented. Hidden surface removal is too slow for real time response, but is a useful tool for clarifying confusing wire-frame images.

10.1.7 User interaction. The response time of AUTOWELD is fast enough to avoid unacceptable delays in an interactive environment. AUTOWELD responds to user commands within one frame update. The delay depends on the current mode and the scene complexity, but is usually less than 2 seconds. The maximum delay for a moderately complex scene is about 10 seconds if shaded solids are being used. If desired, the speed can be increased by upgrading the hardware with the addition of a floating point processor and the latest CPU board.

AUTOWELD is designed so that the response to a pick with the mouse is immediate. This insures that the cursor doesn't move during the delay between the user pick and its processing by AUTOWELD.

AUTOWELD gives the user great flexibility and control in developing a welding program. Verification and modification of the program at any point are easily accomplished.

10.2 Suggestions for Further Work

The work described in this report constitutes a major building block for a comprehensive off-line programming system. To develop AUTOWELD into a complete system, the following tasks can be identified:

1. CAD Interface

   AUTOWELD must be interfaced to a CAD data base.

2. Expert Welder

   A comprehensive and practical expert welder system must be developed and integrated into AUTOWELD.

3. Sensory Feedback

For successful execution of the nominal paths specified by AUTOWELD, a real-time seam tracking system should be incorporated into the welding workstation.

4. Testing and Application

The system must be tested and developed by actually welding using a real workstation.

The problem of path planning to avoid collisions using redundancy has not been solved. This is a difficult academic problem of current interest in robotics. A great deal of academic effort remains before a satisfactory general technique can be developed which is efficient and flexible.

A successful path planning algorithm will most likely require a great deal of processing, precluding its implementation in an interactive environment. In principle, it is possible that AUTOWELD will develop into a completely automtic system which requires no interactive user input. In that case, the execution speed of AUTOWELD will be less critical as long as a feasible program is generated in a reasonable time at a reasonable cost.

11. REFERENCES

BAER,A., EASTMAN,C., HENRION,M., "Geometric Modelling: a Survey", Computer-aided Design, Vol.11,No.5, 1979, pp.253-72.

BENATI,M., MORASSO,P., TAGLIASCO,V., "The Inverse Kinematic Problem for Anthropomorphic Manipulator Arms", Transactions of the ASME, Vol.104, March 1982, pp.110-113.

BOYSE,J.W., "Interference Detection Among Solids and Surfaces", Comm. of the ACM, Vol.22, No.1, Jan.1979, pp.3-9.

BRADY, HOLLERBACH, JOHNSON, LOZANO-PEREZ, MASON, eds., Robot Motion: Planning and Control, MIT Press, 1982.

BRAID, I.C., "The Synthesis of Solids Bounded by Many Faces", Graphics and Image Processing, Vol.18, No.4, April 1975, pp.209-216.

BROOKS, R., "Planning Collision-Free Motions for Pick-and-Place Operations", International Journal of Robotics Research, Vol.2, No.4, 1983, pp.19-44.

BUCHAL,R.,SASSANI,F.,CHERCHAS,D.,DUNCAN,J.P., Development of an Automatic Welding System for a Welding Robot: Phase I, University of British Columbia Department of Mechanical Engineering Computer-Aided Manufacturing and Robotics Laboratory (CAMROL), CAMROL R.84-1, February 1984.

DENAVIT,J., HARTENBERG,R.S., "A Kinemtic Notation for Lower-Pair Mechanisms Based on Matrices", Journal of Applied Mechanics, Vol.22, No.1, 1955, pp.215-221.

FOLEY,J.D., VAN DAM,A., Fundamentals of Interactive Computer Graphics, Addison-Wesley, 1982.

FOURNIER,A., KHALIL,W., "Coordination and Reconfiguration of Mechanical Redundant Systems", Proc. International Conference Cybernetics Society, 1977.

FRANKLIN,W.R., "Efficient Polydedron Intersection and Union", Graphics Interface '82, pp.73-80.

GAGLIO,S., MORASSO,P., TAGLIASCO,V., ZACCARIA,R., "Computation of Inverse Kinematics and Inverse Dynamics in Manipulator Arm Control", Proc. 11th Int. Symp. on Industrial Robots, Tokyo, 1981.

HOSAKA,M., KIMURA,F., KAKISHITA,N., "A Unified Method for Processing Polyhedra", Information Processing 74, 1974, pp.768-772.

KALAY,Y.E., "Interactive Shape Generation and Spatial Conflict Testing", 18th Design Automation Conf., 1981, pp.75-81.

KIRCANSKI,M., VUKOBRATOVIC,M., "Trajectory Planning for Redundant Manipulators in the Presence of Obstacles", 5th CISM, 1984, pp.43-50.

KONSTANTINOV,M., MARKOV,M., "Discrete Positions Method in Kinematics and Control of Spatial Linkages", Mechanism and Machine Theory, Vol.15, 1980, pp.47-60.

KONSTANTINOV,M., GENOVA,P., ZAHARIEV,E., "Direct Kinematic Control of Industrial Manipulators and Robots", Proc. 11th Int. Symp. on Industrial Robots, Tokyo, 1981.

KONSTANTINOV,M., MARKOV,M., NECHEV,D., "Kinematic Control of Redundant Manipulators", Proc. 11th Int. Symp. on Industrial Robots, Tokyo, 1981.

KONSTANTINOV,M., PATARINSKI,S., "A Contribution to the Inverse Kinematic Problem for Industrial Robots", Proc. 12th Int. Symp. on Industrial Robots, Chicago, 1983.

LEE,C., ZIEGLER,M., "A Geometric Approach to Solving the Inverse Kinematics of PUMA Robots", Proc. 13th Int. Symp. on Industrial Robots, Chicago, 1983.

LIEGEOIS,A., "Automatic Supervisory Control of the Configuration and Behaviour of Multi-Body Mechanism", IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-7, No.12, 1977, pp.868-871.

LOZANO-PEREZ,T., "Spatial Planning: A Configuration Space Approach", IEEE Transactions on Computers, Vol.C-32, No.2, 1983, pp.108-120.

MARUYAMA,K., "A Procedure to Determine Intersections Between Polyhedral Objects", Int. J. of Computer and Information Sciences, Vol.1, No.3, 1972, pp.255-264.

MILENKOVIC,V., HUANG,B., "Kinematics of Major Robot Linkage", Proc. 13th Int. Symp. on Industrial Robots, Chicago, 1983.

PAUL,R., SHIMANO,B., MAYER,G., "Kinematic Control Equations for Simple Manipulators", IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-11, No.6, June 1981, pp.449-455.

PAUL,R., Robot Manipulators: Mathematics, Programming and Control, MIT Press, 1981.

REQUICHA, A., "Representations of Solids: Theory, Methods and Systems", Computing Surveys, Vol.12, No.4, 1980, pp.437-465.

ROGERS, D.F., Procedural Elements for Computer Graphics, Mcgraw-Hill, 1985.

SCHWARTZ,J., "Finding the Minimum Distance Between Two Convex Polygons", Information Processing Letters, Vol.13, No.4,5, 1981, pp.168-170.

VUKOBRATOVIC, M., KIRCANSKI,M., "Method for Optimal Synthesis of Manipulation Robot Trajectories", ASME Journal of Dynamic Systems, Measurement and Control, Vol. 104, No.2, June 1982, pp.188-193.

WESLEY,M., LOZANO-PEREZ,T., LIEBERMAN,L., LAVIN, M., GROSSMAN,D., "A Geometric Modelling System for Automated Mechanical Assembly", IBM Journal of Research and Development, Vol.24, No.1, 1980, pp.64-74.

WHITNEY,D., "The Mathematics of Coordinated Control of Prosthetic Arms and Manipulators", ASME Journal of Dynamic Systems, Measurement and Control, Dec.1972, pp.303-309.

APPENDIX I - THE FACE SPLITTING INTERFERENCE DETECTION METHOD

This appendix contains documentation of the face splitting interference method developed early in the contract period. The data structures and software have evolved since the state described here so some inconsistencies may exist between this documentation and later documentation describing the current system.

I.1 DATA STRUCTURE

The geometric data base is comprised of two parts; the invariant topology and the specific geometry. For a system of rigid bodies subjected only to rigid body motions, the topology is invariant but the geometry relative to a world coordinate frame changes. As well as the topology, we maintain both the initial static geometry and the current transformed geometry.

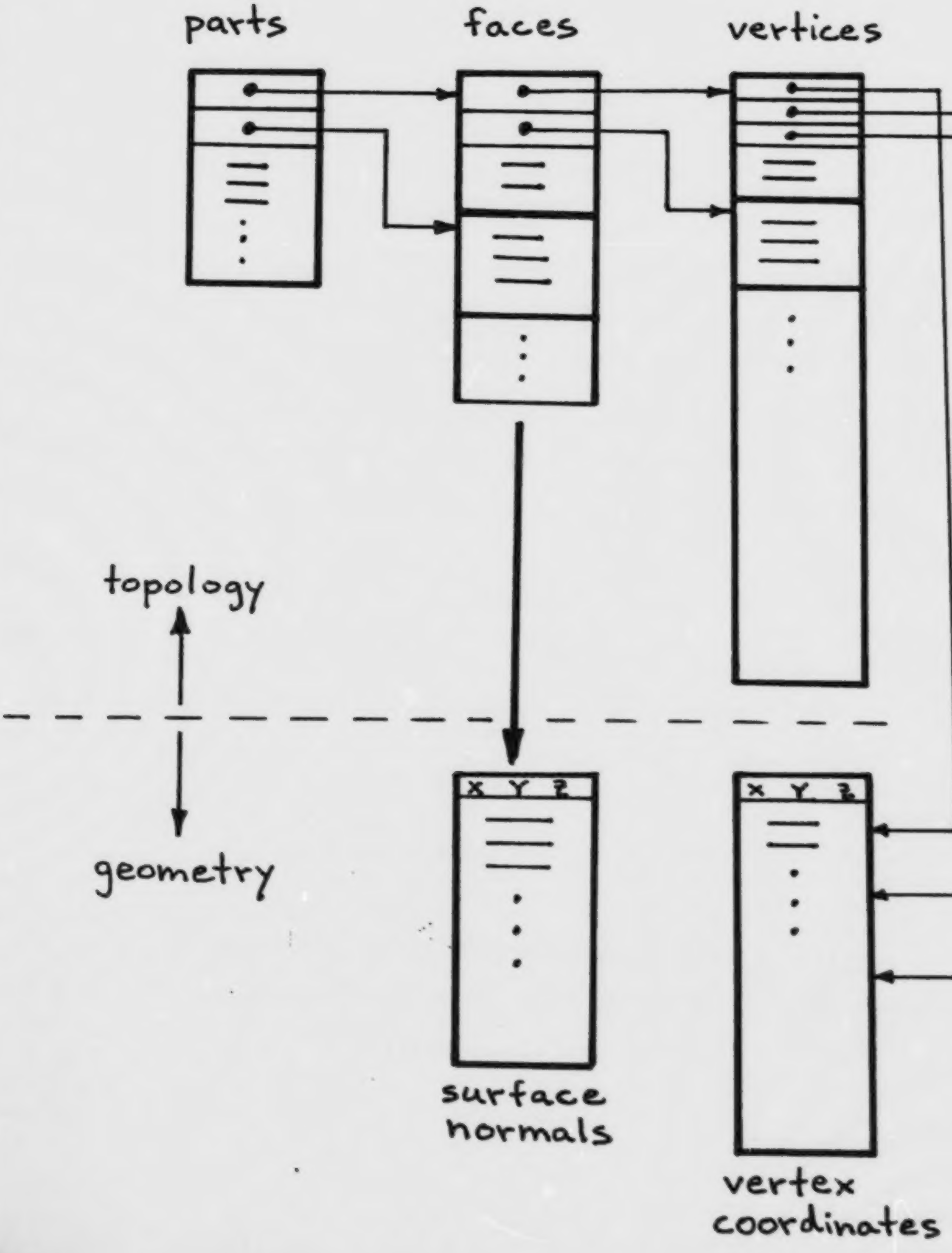
I.1.1 Static Data Structure. The geometric data base consists of a set of common arrays and parameters which are initially read from files. The structure of the data base is shown in Figure I.1.

I.1.1.1 Global Arrays. The data arrays and their contents are as follows:

a. PARTS: the list of parts (separate objects) in the workstation.

PARTS(1,i)	index of the first face of part(i) in FACES.
PARTS(2,i)	the number of faces of part(i).

Figure I.1- Geometric Data Structure


parts
faces
vertices
topology
geometry
X Y Z
X Y Z
surface normals
vertex coordinates

PARTS(3,i)	index of the first vertex number of part(i) in VERTS.
PARTS(4,i)	the number of vertices of part(i).

b. FACES: the list of faces.

FACES(1,i)	index of the first vertex of face(i) in VERTS.
FACES(2,i)	the number of vertices of face(i).

c. VERTS: the list of vertex numbers for each face.

d. VCOORD: the coordinates (x,y,z) of the vertices.

e. FNORMS: the face normals corresponding to the faces in FACES. The normals are stored as the true surface normal plus the coordinates of the first vertex of the face. This allows preservation of the normals after rotations and translations.

Besides these arrays which are read in from files, several more arrays are deduced from them:

a. OVERTS: a list of vertex adjacencies.

OVERTS(1,i)	index of the first face adjacent to vertex(i) in FACES2.
OVERTS(2,i)	the number of adjacent faces.
OVERTS(3,i)	index of the first adjacent vertex in AJVERT.
OVERTS(4,i)	the number of adjacent vertices.

b. FACES2: a list of face indices pointing to FACES.

c. AJVERT: a list of vertex indices pointing to VCOORD.

I.1.1.2 Global Parameters. A set of global parameters are also read from the data file. These parameters are:

a. NPT: the number of parts in PARTS.

b. NFAC: the number of faces in FACES.

c. NVL: the number of vertex indices in VERTS.

d. NV: the number of vertices in VCOORD.

Also, the following global parameters are deduced:

a. NVL2: the number of vertices in OVERTS.

b. NFACE2: the number of faces in FACES2.

c. NVADJ: the number of vertices in AJVERT.

I.1.2 Dynamic Data Structures. As well as the static data structure above which never changes, a dynamic data structure is also maintained. The transformed data is obtained by multiplying elements of the initial geometry coordinate arrays by the current transformations. This data structure contains the current transformed geometric data in the following arrays:

a. TCOORD: a transformed vertex coordinate list. The format is the same as for VCOORD.

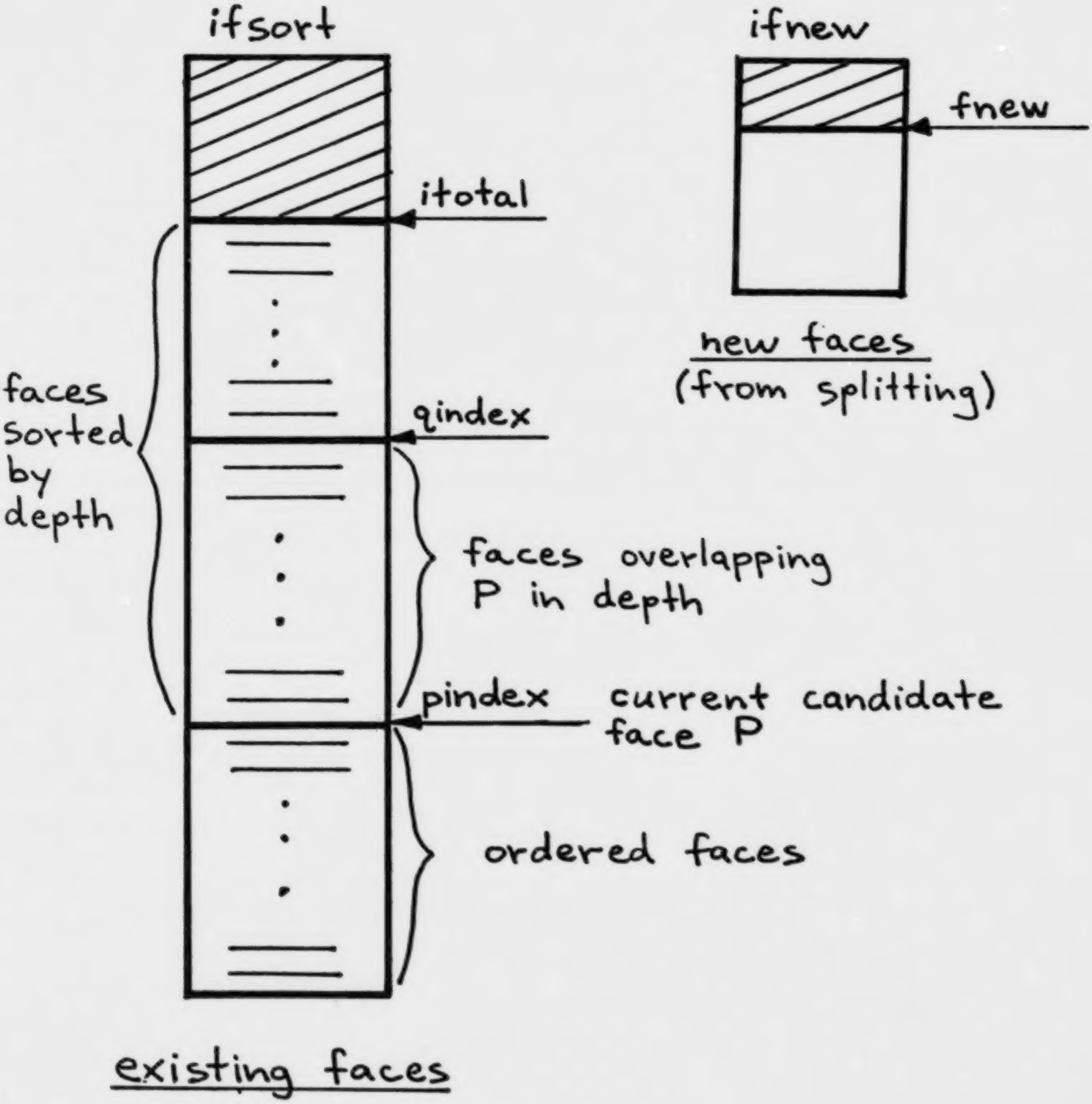
b. TNORMS: a transformed face normals list. These are the true normals.

c. SCOORD: the integer screen coordinates of the vertices including the perspective depth.

I.1.3 Data Structure for Interference Detection. A dynamic data structure is created by the hidden surface removal and interference detection software each time a new frame is processed. This data structure is shown in Figure I.2. The data consists of the following:

a. IFSORT(3,n): this is the face list used by the algorithm. The dimension n must be chosen to be greater than the largest anticipated

Figure I.2- Dynamic Data Structure


ifsort
itotal
qindex
pindex
current candidate face P
faces sorted by depth
faces overlapping P in depth
ordered faces
existing faces
ifnew
fnew
new faces
(from splitting)

number of faces. IFSORT is manipulated so that only one list is needed. The current head of the list contains the candidate face P, which is tested against the remaining unordered faces in the list. Face P is either output to an ordered list and removed from the unordered face list, or it is reinserted into the unordered face list and another face becomes face P.

IFSORT(1,i)	This is the number of the part to which the face belongs.
IFSORT(2,i)	This is the index of the face in the array FACES.
IFSORT(3,i)	This is a special status field. It can have one of the following values: 0 - initial value 1 - face P is possibly intersected by this face. 2 - this face has been split and must be discarded from the list. 3 - this face is outside of the test polyhedron. 4 - the current candidate face P is split by the plane of this face. 5 - this face is split by the plane of face P. 6 - face P and this face split each other.

b. IFNEW(3,n): this is an array like IFSORT which temporarily stores new face fragments which result from face splitting.

c. BOX(6,n): contains bounding boxes corresponding to the entries of FACES. The boxes are stored as (xmin, xmax, ymin, ymax, zmin, zmax).

d. ITOTAL: the current end of the list IFSORT.

e. FNEW: the current end of the list IFNEW.

f. PINDEX: This is the index of the current head of the face list. Each time the current candidate face P passes all of the hidden surface tests, PINDEX is incremented by one. In effect, face P is removed from the unordered face list and added to the ordered output face list. Only one array is used, and no sorting is needed.

g. QINDEX: The unordered face list IFSORT is always ordered by maximum screen depth (maximum Z value). An initial Z extent comparison determines the set of faces which overlap the current candidate face P in depth. The parameter QINDEX is the index of the last face in this set. No faces with a higher index need be considered further.

I.2 DESCRIPTION OF MATHEMATICAL METHODS

A number of simple mathematical methods are used in the geometric calculations done by the software package. Some of the methods are described in this section.

I.2.1 Surface Shading. A simple surface shading model is used to instill some visual realism. The method consists of taking the dot product of the line of sight and the surface normal of a face. The resulting product is used to generate a pointer into a gray scale. A normalized dot product of -1 (direct illumination) gives white, and a product of zero gives black.

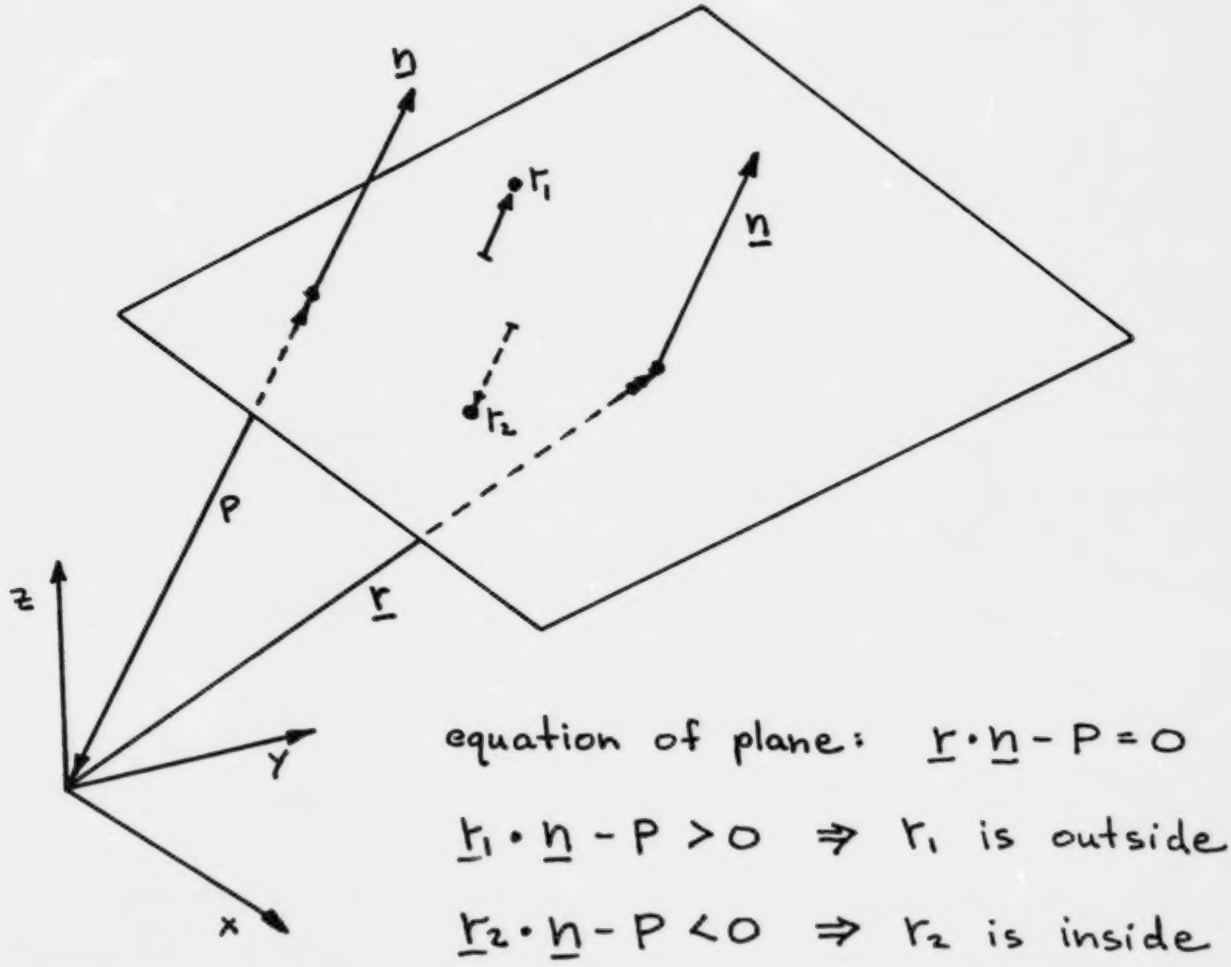
In the current implementation, 16 shades of gray are defined. This model simulates simple diffuse reflection with the light source directly behind the viewer.

In our method, a single shade is chosen for each face. In the case of perspective projection, the dot product (and thus the shade) varies over a face. This discrepancy becomes apparent when the algorithm assigns different uniform shades for different fragments of the same face.

I.2.2 Testing for Possible Face Intersection. Two faces cannot possibly intersect if neither one is intersected by the plane of the other. This is true if each face is wholly on one side of the plane of the other. To test this, the surface normal and distance from the origin are found for the test plane. This allows us to specify its vector equation. The vertices of the test face are inserted into this equation, and the sign of the result indicates which side they are on. This is illustrated by Fig. I.3.

I.2.3 Testing for Overlap in Screen Coordinates. The hidden surface removal algorithm requires that we test if the projections of two faces are disjoint. To do this, we first transform the vertices of the faces into screen space. The problem can now be solved in 2-D. The coordinates are in integer screen coordinates, so integer arithmetic is used. Two convex polygons are disjoint if a line defined by any edge of either one separates the polygons into two disjoint half-planes. A convex polygon always lies on one side of the line defined by any of its edges. Thus, a second polygon is disjoint if all of its vertices lie on the side opposite

Figure I.3- Definition of a Plane


$\underline{n}$
r_1
$\underline{n}$
r_2
P
$\underline{r}$
z
Y
x

equation of plane: $\underline{r} \cdot \underline{n} - P = 0$

$\underline{r}_1 \cdot \underline{n} - P > 0 \Rightarrow r_1$ is outside

$\underline{r}_2 \cdot \underline{n} - P < 0 \Rightarrow r_2$ is inside any vertex of the first. This is illustrated in Fig. I.4. The test can stop as soon as this condition is found. If no edges of either polygon satisfy this condition, then the polygons overlap. As opposed to the methods of edge intersection and vertex inclusion, this method allows us to expolit the coherence of convex polygons to reduce the number of calculations. Some of these properties can be stated as:

a. If a vertex of a polygon is closer to a line than either of its neighbour vertices, then it is the closest vertex of the polygon. No other vertices need to be tested. In Fig. I.5, we find that vertex 2 is closer than 1 or 3. Therefore, it must be closer than vertex 4, which need not be tested.

b. If an edge of a polygon yields a larger minimum distance to a second polygon than either of its neighbouring edges, then it is the best separating edge and no other edges need to be tested. This is shown in Fig. I.6. Edge 2 is a better separating edge than its neighbouring edges 1 and 3 because

$$(-d2) > (-d1) \text{ and } (-d2) > (-d3).$$

Thus -d1 is a maximum and we need not test edge 4.

These properties, which can be extended to 3-D polyhedra, allow many calculations to be eliminated. This should prove useful when dealing with convex polygons (or polyhedra) with many edges (or faces).

I.2.4 Intersection of a Polygon by a Plane. If a face, represented by a convex polygon, is intersected by a plane, then exactly two of its edges

Figure I.4 - Separating Line between Disjoint Polygons

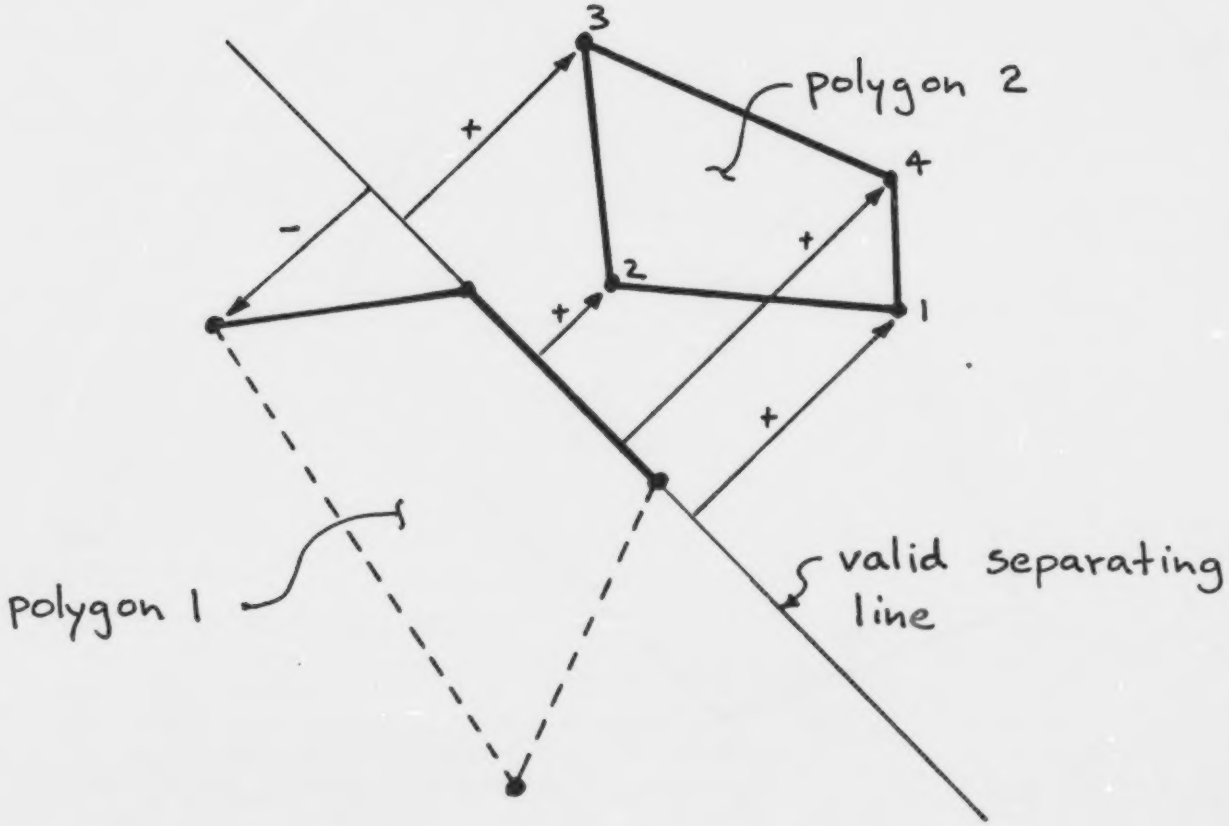

3
polygon 2
4
+
−
+
2
+
1
+
valid separating line
polygon 1

Figure I.5- Best Separating Line

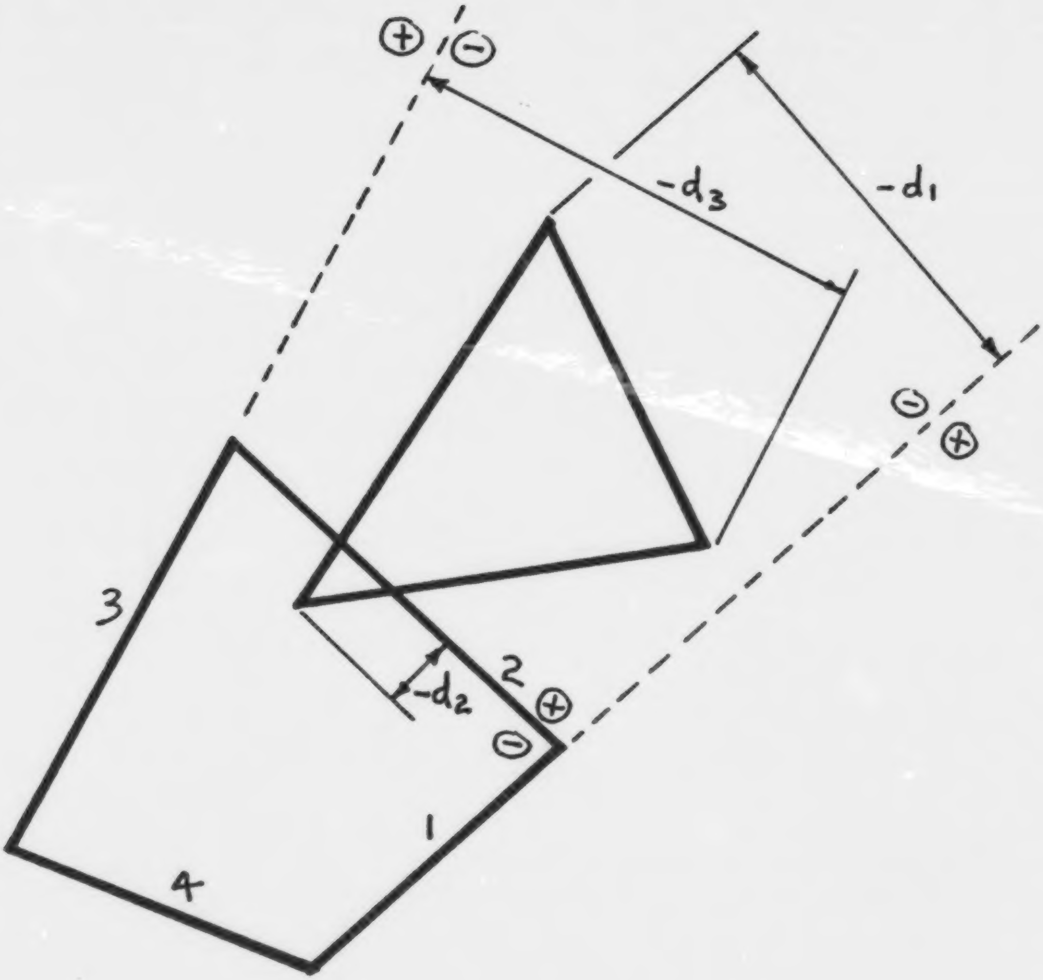
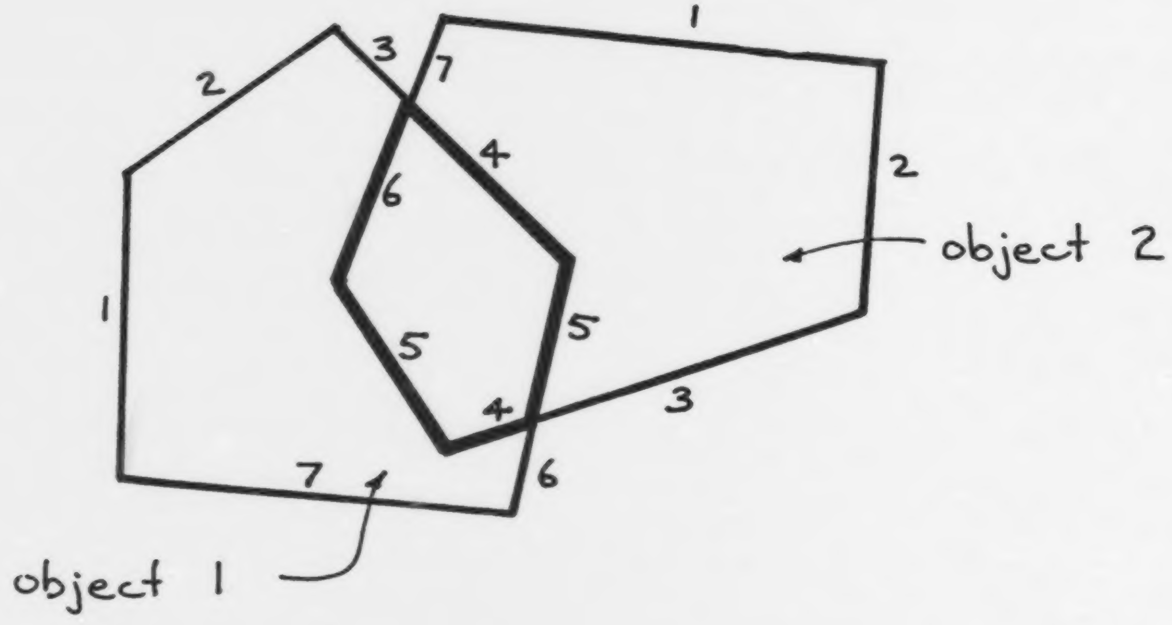

⊕
⊖
$-d_3$
$-d_1$
⊖
⊕
3
2
⊕
⊖
$-d_2$
1
4

Figure I.6- Determination of Interior Faces


1
2
3
4
5
6
7
object 1
1
2
3
4
5
6
7
object 2

intersect the plane. An edge intersects a plane if its ends are on opposite sides of the plane, or if its first vertex lies on the plane. Vector equation of an edge with vertices V1 and V2:

$$\underline{r} = \underline{V}1 + (\underline{V}2 - \underline{V}1)\, t, \quad 0 \le t \le 1 .$$

Vector equation of a plane:

$$\underline{r} \cdot \underline{n} - P = 0 .$$

Combining, we get

$$t = \frac{P - \underline{V}1 \cdot \underline{n}}{(\underline{V}2 - \underline{V}1) \cdot \underline{n}}$$

or,

$$t = \frac{P - \underline{V}1 \cdot \underline{n}}{(\underline{V}2 \cdot \underline{n}) - (\underline{V}1 \cdot \underline{n})} .$$

The edge intersects the plane if $0 \le t < 1$. Thus, for the special case of a vertex lying on the plane, only one of the two edges will yield an intersection point.

I.2.5 Determination of Interior Faces. If we know that no faces intersect each other, then to determine whether a face lies inside or outside of another face, we need only test a single vertex of that face against the plane of the other face. We know that all of the vertices must be on the same side.

We can find the set of faces defining the volume of intersection of two polyhedra in two steps:

1. Find all of the faces of polyhedron 1 which lie inside all of the faces of polyhedron 2.
2. Now find all of the faces of polyhedron 2 which lie inside all of the faces found in 1.

Note that the second step is considerably faster than the first.

The method is illustrated by Figure I.6. First, faces 4 and 5 of object 1 are found to be inside object 2. Then, faces 4, 5 and 6 of object 2 are found to be inside object 1. Note that faces 4, 5 and 6 of object 2 are inside object 1 if they are inside faces 4 and 5 of object 1.

I.3 Software Structure

The software structure is shown graphically as follows. Each level of indentation represents another level of subroutine nesting.

```
MAIN
    RDMODL
    TRAN
    SORT
        SCREEN
    HSURF
        HTESTS
            PLSIDE
            OVRLAP
        SORT2
            PLSIDE
        SWAP
        SPLIT
            SCREEN
        RESORT
    INTERF
```

DRMODL
    DRPOLY
DRINT
    DRPOLY

I.4 List of Subroutines

MAIN: This is the main interactive graphics program.

calls - DRINT, DRMODL, HSURF, INTERF, RDMODL, SORT, TRAN

DRINT: This subroutine draws the volume of intersection.

called from - MAIN

calls - DRPOLY

DRMODL: This subroutine draws the objects in the world model.

called from - MAIN

calls - DRPOLY

DRPOLY: This subroutine draws filled polygons.

called from - DRINT, DRMODL

calls - none

HSURF: This subroutine performs the hidden surface removal and interference detection.

called from - MAIN

calls - HTESTS, RESORT, SORT2, SPLIT, SWAP

HTESTS: This subroutine performs a hierarchy of tests on a candidate face.

called from - HSURF

calls - OVRLAP, PLSIDE

INTERF: This subroutine finds the faces which belong to the intersection volume.

called from - MAIN

calls - none

OVRLAP: This subroutine tests if two faces overlap on the screen.

called from - HTESTS

calls - none

PLSIDE: This subroutine tests two faces to see if either is intersected by the plane of the other.

called from - HTESTS, SORT2

calls - none

RDMODL: This subroutine reads the geometry data base from a file.

called from - MAIN

calls - none

RESORT: This subroutine generates the bounding boxes for new face fragments, and inserts the new face fragments into the face list.

called from - HSURF

calls - none

SCREEN: This subroutine converts object space coordinates into screen coordinates, including perspective depth. This emulates the IRIS routine SCREEN.

called from - SORT, SPLIT

calls - none

SORT: This subroutine performs an initial depth sort of the face list and finds the face bounding boxes.

called from - MAIN

calls - SCREEN

SORT2: This subroutine finds mutually intersecting faces.

called from - HSURF

calls - PLSIDE

SPLIT: This subroutine splits faces.

called from - HSURF

calls - SCREEN

SWAP: This subroutine inserts face P into the face list and replaces it with another candidate face.

called from - HSURF

calls - none

TRAN: This subroutine transforms the geometry of a specified part by the current transformation matrix.

called from - MAIN

calls - none

APPENDIX II - DUNCAN INTERFERENCE DETECTION METHODS

AUTOMATIC TESTING FOR COLLISION IN ROBOTICS

by

J.P. Duncan

Summary

Two alternative methods are presented for automatically detecting and signalling collision between any single link of an articulated robotic arm and fixed solid objects in the working space in which the arm has a preplanned prescribed path.

Method I is based on a so-called TETRAHEDRAL CONCEPT. Method II may be described as the EDGE-FACET INTERSECTION METHOD.

A fixed object is represented by an irregular polyhedron specified in terms of the coordinates of a few of the infinity of points of its bounding surface. Such points, linked by straight lines, are 'organized' by way of a plane "Mercator's projection" of the spatial surface of such an object to provide an 'indicial grid' - a topologically rectangular array - which allows a computer to find and manipulate the coordinates of the vertices of the polyhedron in any transformed position in space. Such an array is simular to that used in the author's system, POLYHEDRAL NC, (Reference 4). It recognizes the fact that surface is a two-dimensional entity set in three dimensional space as enunciated by Gauss. In a generalized way such a surface and its plane projection are characterized by 'latitude' and 'longitude'.

A single link of an articulated robotic arm may likewise be represented approximately by an irregular polyhedron. A right parallelepided, having eight vertices and located instantaneously in any arbitrary position in space, also specified by a "Mercator's projection", is used to represent the link for purposes of illustrating the working of both methods.

Method I is suited to detection of collision of the arm with a complex, convex or cancave polyhedron requiring 'dense' point-specification. It tests to see whether volumetric spaces enclosed by object- and link-surfaces respectively co-exist in some localized sub-region of coordinate space, thus indicating that the arm and object will collide in their preplanned relative spatial trajectory.

Method II is guaranteed to detect any intersection of the edges of the two polyhedra with faces (facets) of the other within the length of the edge and within the triangular boundary of a plane facet. The method is general and inclusive of all conceivable configurations but even to check for collision between two sparsely specified eight-vertexed general polyhedra, some 400 - 500 similar computations must be made for a full examination of the general case. Yet by adopting certain acceptable measures and by using the principle of 'boxing' as an initial 'filter', the time required for a reasonably capable computer to examine and report on a series of relative locations from a preplanned trajectory is quite small.

In Method I a tactic for avoiding collision by variation of the preplanned trajectory is suggested but not developed. Links of an articulated arm in any instantaneous configuration require checking one by one and at each stage of the trajectory. If collision is detected and a new acceptable location for the colliding link has been found for a given instant in the trajectory, the inverse kinematic problem of the complete articulated robotic arm must be solved to relocate the other links of the arm. These must connect the relocated link to the workhead at one end and to the foundation at the other according to the degrees of freedom available. The procedure must be repeated for each relocated link and all links of the arm must be tested and if necessary relocated in progression to give general assurance of non collision.

Examples of the application of both Method I and Method II to arbitrary, static relative relationships of an arm and an object are given and demonstrable for an assumed trajectory on terminals driven by VAX or IRIS computers. Listings of both BASIC and FORTRAN programs used for such demonstrations are appended.

Robot Arm - Workpiece Configuration

A robot arm consists of an articulated open kinematic chain with active pairs (joints) connecting a base or foundation with a workhead, such as a welding unit with its projecting welding rod, as shown in Figure 1.

Each link of the arm will be represented approximately by a general polyhedron defined in terms of the coordinates of its vertices. Such an arm may be of any general shape so defined but it will be assumed here that a typical link may be represented by a right parallelepiped having eight vertices, six faces and twelve edges. These numbers agree with Euler's rule for convex polyhedra:

Vertices + Faces - Edges = 2

8 + 6 - 12 = 2

A general eight-vertexed polyhedron having all triangular faces (the plane parallelograms of a parallelepiped become two triangles joined on the diagonal of the faces) has eight vertices, twelve faces and 18 edges. Again, Euler's rule applies:

8 + 12 - 18 = 2

A parallelepiped-link is shown in Figure 1, one of several which make up the articulated arm.

Also shown in Figure 1 is a general workpiece, a concave polyhedral surface bounding a solid volume in the form of a finial (end-piece), the pièce de resistance of the skilled sheet-metal worker.

Figure 2 shows how such a worker 'develops' such a surface-shape using the concept of triangular faces or facets, contiguous across straight line segments joining adjacent facets at folds along these segments.

This finial, defined in terms of vertices numbered 1 - 16 for a typical 'petal', is a concave polyhedron. The test of concavity is ability to draw a straight line from a point on the surface cutting the surface elsewhere at two other points. If a polyhedron is convex, such a line would cut the surface elsewhere at only one other point.

That portion of the finial bounded by the lower region and the cone whose slant heights are tangents to the finial at the circle of inflexion at A-A is convex. Any point within this conical-topped volume will form with any and all finial facets an irregular tetrahedron having four vertices, four faces and six edges. All such tetrahedra taken together nest exactly within the volume bounded by the finial-polyhedral surface. (To close the finial surface, N and S 'poles' are selected near the ends and are joined by straight lines to points on the nearest adjacent 'circle of latitude').

The TETRAHEDRAL CONCEPT as developed and applied below rests on the above geometrical facts.

If a concave polyhedral workpiece has the character of a solid built of convex volumes as in Figure 3, it may be treated as a compound of several such convex polyhedra and each one can be processed as now to be explained for a single polyhedron.

If the spatial configuration of the articulated robot arm is specified from independent considerations at each instant of time in a space-time trajectory, the test of collision is a static geometrical problem at each of a series of incremental times.

The general relationship of a densely specified finial-workpiece and a sparsely specified parallelepiped-arm is shown in Figure 4 in a non-interfering (non-colliding) relationship. By varying this relative location of arm and workpiece, the structure and working of the algorithms used in Method I based on the TETRAHEDRAL CONCEPT will be explained and demonstrated.

The spatial relationship of two sparsely-specified eight-vertexed right parallelepipeds (or general eight-vertexed, triangular-faced polyhedra) is shown in Figure 5. This has been selected to demonstrate the generality and complete adequacy of the EDGE-FACET INTERSECTION METHOD, Method II, for the case for which Method I has limitations unless modified by introducing redundant data, a simple measure which keeps the two methods competitive.

Indicial Grid Representation of Polyhedral Surfaces

In both Methods I and II the polyhedral boundary surface of a general object is represented in a manner analogous to the representation of the Planet Earth's surface by a Mercator's plane projection as in Figures 6 and 7.

If such a general arbitrary surface is closed (as for a potato, for instance, Figure 8) or as for a solid built from two cubes (Figure 3), arbitrary lines of 'latitude' and 'longitude' can be drawn on such surfaces and their points of intersection used as vertices of a polyhedral approximation to the surface. This representation conforms in a general way with Gaussian, two-parameter representation of surface residing in three-dimensional Cartesian space.

Figure 9 shows how the finial can be thought of as a 'distorted Planet Earth' whose petal-boundaries form generalized 'lines of longitude' and whose plane sections parallel to the base form 'lines of latitude'. To close the surface we invent a 'N pole' and a 'S pole' and join them to points on 'arctic and antarctic circles' to complete the 'lines of longitude'. The lateral ends of the unwrapped and distorted plane representation of the surface (Mercator's projection) can be thought of as the two representations of the 'international date line'. The addition of a 'Greenwich meridian' and an 'equator' completes the analogy.

Figure 5 shows how each of two parallelepipeds in arbitrary relative locations can be represented by two 'Mercator's projections' (I,J) and (K,L); (longitude/latitude).

In all these cases the coordinates of intersections of 'curves' of 'longitude' and 'latitude' are treated as vertices of approximating spatial polyhedron, each having coordinates x, y, z in Cartesian space.

Thus any vertex at indicial grid point (I,J) or (K,L) has coordinates $x_{I,J}$, $y_{I,J}$, $z_{I,J}$ and all such sets may be stored for recall in terms of I and J. The whole of the stored set of coordinates may if desired be homogeneously transformed and then recalled again in terms of I and J. Once the coordinates have been input for any one position of an object or arm, all further operations are managed in terms of I and J only.

Hence, the coordinates of vertices and dependent facets and the edges of both workpiece- and arm-polyhedra in any positions in space can be readily recalled for computer processing by algorithms designed to test whether collision is occuring. Further, this proves to be possible at acceptably high speeds of computation and indication by signalling.

Tests for Collision

One test is to determine whether any part of the volume enclosed by one polyhedral surface coexists with that enclosed by the other. This is the basis of Method I using the TETRAHEDRAL CONCEPT.

Another test is to see if any of the edges of one polyhedral surface - say the arm - intersect any face (or facet) of the other - the workpiece - within the triangular boundary of the plane facet and between the ends of the edge line-segment. For the test to be conclusive the edges of the workpiece must be tested for intersection with the faces of the arm: the test must be reciprocal. This approach is the basis of the EDGE-FACET INTERSECTION METHOD - Method II.

Theory of Method I

If a reference point is any point within a convex polyhedron or a suitably chosen point within a concave polyhderon it may be joined by straight lines to each and every vertex of the polyhedron (with poles as explained above) to form a set of nesting tetrahedra occupying exactly the volume enclosed by the polyhedral surface.

Let a typical facet be ABC and the common point D refer to Figure 10 where point D is conveniently chosen as the origin.

The volume of irregular tetrahedron ABCD is given in terms of the coordinates of its vertices by the following fraction of a determinant:

$$1/6 * \begin{vmatrix} x_A & y_A & z_A & 1 \\ x_B & y_B & z_B & 1 \\ x_C & y_C & z_C & 1 \\ x_D & y_D & z_D & 1 \end{vmatrix}$$

The centroid G of the triangle ABC has coordinates:

$$x_G = 1/3\ (x_A + x_B + x_C)$$
$$y_G = 1/3\ (y_A + y_B + y_C)$$
$$z_G = 1/3\ (z_A + z_B + z_C)$$

The centroid, T, of the tetrahedron is at a height equal to 1/4 of the height of the vertex D above the base ABC. For simplicity take the origin, point 0,0,0 as point D, as in Figure 10. Let v represent the volume of the tetrahedron as determined from the above determinant. Then the moments, m_i, of this volume about the origin are:

$$m_x = v \cdot x_T$$

$$m_y = v \cdot y_T$$

$$m_z = v \cdot z_T$$

where

$$x_T = 3/4\ x_G$$

$$y_T = 3/4\ y_G$$

$$z_T = 3/4\ z_G$$

The total volume of the polyhedron is $\Sigma v = V$. The total moment of all tetrahedra forming the polyhedron is

$$\Sigma\, m_i = M_i \quad i = x,y,z$$

The coordinates of the centroid, P, of the polyhedron as a whole are

$$M_i \,/\, V \quad i = x,y,z$$

The above computations have been applied to the finial with its axis coincident with the z coordinate axis and its base in the xy coordinate plane. This position and the location of its centroid, P, is indicated in Figure 9. Application of homogeneous transformation or its equivalent using Euler angles enables the polyhedron and its centroid to be represented in any part of coordinate space as in Figure 4.

The parallelepiped

Clearly, the parallelepiped is a convex polyhedron, the coordinates of whose few (8) vertices can be organized in an I,J array as in Figure 11. When defined in a paraxial orientation centred on the origin its volume and centroid location are known through simple calculation. The location of all its vertices (corners) and its centroid in any displaced position may again be found by homogeneous transformation. The displaced parallelepiped and its surrounding 'box' are shown in such displaced positions in Figures 4 and 12. In Figure 12 we have plotted all edges and vertices of the parallelepiped and its 'box' but to avoid unnecessary confusion of line, have left the polyhedron in an undisplaced location and have plotted its front side only.

Relative location of polyhedra

Recall that the polyhedron -- the finial -- is meant to represent a 'workpiece' and the parallelepiped a link of the 'arm' of a robot. We wish to know if, during a proposed relative motion of these two representative elements, any part of their individual volumes coexist in some region of coordinate space.

Let the finial remain in its defined position (representing the 'workpiece') and move the parallelepiped to various relative locations such as might result in a preplanned robotic action. Recall that the parallelepiped represents any one link of an articulated 'arm'.

In Figure 12, the paraxial 'boxes' do not overlap in the one orthographic projection shown. No interference can occur. Such a situation can readily be checked by a simple computer program which determines the ranges of coordinates of the two polyhedra involved.

In Figure 13, the two 'boxes' overlap in one view as plotted. Interference between polyhedron and parallelepiped will not occur, however, as can be seen from their plotted images within the 'boxes'. The figure makes the point that although the 'boxes' overlap indicating that interference _might_ occur, the two bodies themselves will not do so.

In Figure 14, interference is a distinct possibility, not only because the 'boxes' overlap in one projection but also because the polyhedron and parallelepiped also overlap in one projection. But it is seen from the plan view of this case as in Figure 15, neither the 'boxes' nor the polyhedra overlap in space.

In Figure 16, the two polyhedra and their 'boxes' overlap in two views, Figure 16 being the elevation for this case.

Interference detection and indication

The last few figures convey their message graphically. In automatic interference checking we do not want to plot everything but rather to receive warning messages that a clash will occur if the planned path is followed and, if possible, to modify that path automatically to avoid the clash.

Let the finial be the densely defined polyhedron, its vertices being stored and recalled at will from an I,J array. If any such vertex is inside the parallelepiped the sum of the volumes of the tetrahedra formed by joining all vertices of the parallelepiped to that polyhedral vertex must equal the known fixed volume of the parallelepiped. If outside the sum will be greater. Thus the basis of an algorithm for indicating whether a workpiece vertex is within the bounded volume of an 'arm' link is as follows:

1. Find the bounding 'boxes' for both workpiece and link by a standard FORTRAN routine.

2. Test to see if the two 'boxes' overlap in space.
3. If so, scan the I,J array of vertex coordinates of the workpiece polyhedron and examine every one of them that lies within the 'common box', that part of space which lies in each single 'box'.
4. For each vertex within the 'common box' test as suggested above to see if it lies within the link -- the parallelepiped in this case.
5. If so, mark that vertex by a cross or circle in a graphic plot as has been done in Figure 16. This generates a cloud of points indicating a region of interference.
6. If indication that interference occurs is all that is required, the process can be stopped at the first interfering point found. A bell may be rung or a message printed to show the parametric value of the relative motion at such interference. This leads to a rapid check of a prescribed path of any link of a robot arm without any immediate action to correct such interference.

Avoidance

When such a 'cloud' appears in an examination of a proposed preplanned robotic path, it is suggested that the following tactic of avoidance be adopted.

The centroids of polyhedron and parallelepiped are joined to provide an arbitrary spatial link between the two bodies having a unit vector v as shown in Figure 1. The axis of the parallelepiped, having a unit vector u as in Figure 1, provides another line in space forming with the first line a plane. Hence a triad of axes can be formed by using the normal to that plane as a third axis having unit vector **n** in Figure 1 where **n** = **u** x **v**.

The angle between the two lines **u** & **v** so defined in a plane can readily be found. The axis of the parallelepiped can then be rotated about the normal **n** until the angle referred to is reduced to zero or made 90°.

For a long slender parallelepiped the closest approach will be when the angle referred to is 90°. If rotation to this orientation does not avoid interference by repeated test, the parallelepiped can be 'inched out' in the direction of its long axis by increments until it is clear.

When a satisfactory position for the parallelpiped has been achieved, the other links of the robot 'arm' need to be adjusted to connect the fixed locations of the two end-links of the assembly to the relocated, interfering link. This is called the inverse kinematic problem. It requires that the articulated arm possesses a sufficient number of degrees of freedom.

If and when the above adjustment has been achieved, other links in the articulated 'arm' must be examined; but since the initial 'box' test will prove that by no means all links will be in trouble, no great labour or difficulty should result.

Even if the procedure of avoidance is not pursued, the method outlined could be used as a silent and vigilant monitor of potential interference, to be corrected by whatever other means might be suggested.

Theory of Method II

If workpieces such as combinations of parallelepipeds are defined by their vertices, of which there may be few, we have what might be called a 'sparse' object-specification. A link of a robot arm represented as a parallelepiped is already 'sparse', having only eight vertices. We show below how the I,J indicial grid can be used to determine whether any edge of an 'arm-link' intersects any plane of the workpiece (parallelepiped) as in Figure 5; that is, whether the edge extended intersects these planes within their boundaries and also within the range of the edge which is but a segment of its extended straight line.

To make the treatment more general, the two parallelepipeds in the following analysis have been treated in such a way that the eight corner points may equally well be non-planar in adjacent sets of four. That is, each parallelogram-side may also be substituted by a pair of plane triangles formed by folding the parallelogram along one of its diagonals. In this general specification of a solid polyhedral figure defined by eight corner points or vertices, there may be from 6 - 12 plane faces of 'facets' in each of the two solid figures and up to 18 edges in each. For as stated above, according to Euler's rule for polyhedra:

$$\text{Vertices} + \text{Faces} - \text{Edges} = 2$$

$$8 + 12 - 18 = 2$$

Thus, in the worst general case, any of the 18 edges of one polyhedron may penetrate any one of the 12 faces of the other. In addition, a case may be postulated in which no edge of one parallelepiped intersects a face or facet of the other. Yet part of the volume of one may co-exist with the volume of the other as shown in Figure 17. To check that possibility, the test of the edges of one intersecting the facets of the other must be reciprocated — the data for the workpiece must be exchanged with the data for the arm and the computation repeated. Hence for a complete check of interference there are 2(12 x 18) = 432 checks to be made for the intersection of a line with a plane in the segment and region of interst respectively in each relative postion of the two polyhedra to be considered in a preplanned spatial robotic progress of the arm, the workpiece being fixed.

In the first demonstration below we have treated the data for two right parallelepipeds in one interfering position but have done so such that the data may be varied to create two arbitrary 12-faceted polyhedra. To avoid special variation of the looping algorithms used, we have retained some redundancies. Certain edges are treated twice. We test for edges being parallel to facets and/or having one end coincidentally lying in a facet. Apart from these special cases we find the coordinates of points where each edge of one parallelepiped intersects each plane of the facets of the other and test to see if that intersection point is within the triangular boundaries of the facet in its plane.

We group the facets (of the workpiece) in 'cells', four to a cell, and likewise group the edges of the arm in 'cells', five to a 'cell'. Using four 'cells' in each parallelepiped, there are 320 cases to be tested of which 64 are really redundant, leaving 256 distinct cases.

The reason why this number is greater than the 216 implied above is that in the interest of convenient surface definition the 'top' and 'bottom' faces of each parallelepiped have been divided into four instead of two facets using a common junction called a 'pole'. This adds four facets to the previous total for check against 18 edges giving 72 extra cases. Against this we have used only four edges for the 'top' and 'bottom' faces of the arm-parallelepiped thus reducing the cases to be considered by 32. Hence 40 redundant cases have to be added to 216 to give the figure of 256.

Data specification

The data for both the eight-pointed workpiece (W) and the eight-pointed arm (R) are set up on the basis of a 'Mercator's projection' of an arbitrary closed surface defined by 'latitude' and 'longitude' as explained above. These are denoted by indicial parameters I,J (workpiece) and K,L (arm) and enable a computer to recall coordinates (x,y,z) data-stored at each location. Thus we speak of 'indicial grids'. A given set of initially stored object points can be recalled from any transformed position of the object by reference to this indicial grid.

To complete the analogy with 'Mercator's projection' we 'invent' a North and a South pole at the centre of the 'top' and 'bottom' faces of the parallelepipeds (any point in space would do) at indicial positions J=4, I=1,2,3 and 4; and J=1, I = 1,2,3, and 4. (Similarly for L,K). Note that all points in rows I = 1 have the same coordinates as is also the case for I = 4. (Similarly for K).

The two parallelepipeds defined in a convenient orientation near the origin of coordinates may be 'placed' in Cartesian space by homogeneous transformation or its equivalent. In practice, the workpiece would be fixed and the position of the 'arm' would be determined at various times by some independently planned motion of a robot-arm assembly, our 'arm' being, kinematically speaking, a link of that robot-arm.

Analysis and logic of edge-facet interference

We now refer to line numbers in the accompanying BASIC computer program "EDGE" conceived as an instructive and illustrative report on all the cases that may occur. More has been printed out than would be necessary in a straightforward signalling of interference.

This BASIC program as given with specific data takes 3 minutes and 20 seconds to run on an inefficient 8 bit computer and uses 38K of floppy disc storage. It reports on the 256 cases to be considered in a one-way check (but also reports certain cases twice for reasons explained above). There would be 512 distinct cases to check if the (W) and (R) were exchanged and the computation repeated. This has been done in the FORTRAN development of the same program as described below.

Lines 1-540 receive the data of 16 points in assumed relative positions of two true right parallelepipeds. These have been chosen in positions as shown in Figure 18 which exhibits some special features of relative configuration to enable the generality of the interference checks to be made in the following program designed to illustrate both special and general cases that might be encountered. (A pair of general eight vertexed polyhedra could be analysed by the same program as used here for parallelepipeds without variation; only the input data would be varied.)

Lines 580 - 660 define points A,B,C,D,N,S of a typical workpiece 'cell' centred at J = 2, I = 1,2,3,4 in turn so that a series of sub-routines may be devised in general terms. Planes 1 - 4 per 'cell' are recognized as in Figure 19.

Lines 670 - 1000 find direction cosines of the normal of each of these four typical planes. The equation of each plane is typically in terms of the facet vertices:

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} = 0$$

Direction numbers, AN, BN, CN for the normals to these planes are given by:

$$AN = \begin{vmatrix} y_1 & z_1 & 1 \\ y_2 & z_2 & 1 \\ y_3 & z_3 & 1 \end{vmatrix}$$

$$BN = \begin{vmatrix} x_1 & z_1 & 1 \\ x_2 & z_2 & 1 \\ x_3 & z_3 & 1 \end{vmatrix}$$

$$CN = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

$$PN = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

from which follow the direction cosines, A,B,C with a numeral added to indicate to which of the four typical planes of a 'cell' they refer.

Lines 1010 - 1170 find the direction cosines of the five edges of each 'cell' of the 'arm' in terms of the coordinates of their end-points, XO and XE.

Lines 1173 - 2050 compute the 320 combinations of arm-edges and workpiece facet-planes (including 64 repeated calculations) and print out a report via sub-routines on interferences and special cases. The theory involved here is as follows:

Each facet-plane of the workpiece has the equation:

$$\alpha x + \beta y + \gamma z + p = 0$$

where α, β, γ, direction cosines of the plane-normal, have been calculated.

Each edge has equations:

$$x = x_o + u.l \quad y = y_o + u.m \quad z = z_o + u.n$$

where x_o, y_o, z_o are coordinates of an edge end-point and l,m,n are direction cosines of the edge. u is a parameter, the distance of point x,y,z along the line from x_o, y_o, z_o.

Assume that the edge and the plane intersect at a point in the plane of the facet (extended). The point of intersection must satisfy the equation of the plane. Thus:

$$\alpha\,(x_o - u.l) + \beta\,(y_o + u.m) + \gamma(z_o + u.n) + p = 0$$

or

$$u = -(\alpha x_o + \beta y_o + \gamma z_o + p) \,/\, (\alpha l + \beta m + \gamma n)$$

$$= \text{(NUMerator/DENominator)}$$

If (x_o, y_o, z_o) is coincident with the plane facet extended, then

NUM = 0

If edge (l,m,n) is parallel to the plane (does not intersect it), then

DEN = 0 and u = infinity.

We can then pass on to the next case.

In the general case, using the BASIC program nomenclature:

XI = XO + U.L YI = YO + U.M ZI = ZO + U.N

We must now determine whether the intersecting point is:

a. Between end-point O and E of the edge being tested

b. Within the triangular boundaries of the facet.

These tests are conducted in a series of checks in sub-routines, lines 5000, 6000 and 6200.

The printed output of the above program "EDGE" first lists the data used (not normally necessary to print). It then identifies each of 320 cases for I = 1 to 4 and K = 1 to 4 and indicates 1) coincidence of the first point of an edge, 2) whether an edge is parallel to the facet considered and 3) the coordinates of an intersection point within a triangular facet.

Kinematic Demonstration of Methods I & II

Listings of two FORTRAN 77 programs for revealing the occurrences of collision with workpieces during a postulated space-time trajectory of a parallelepiped-link of a robot arm are appended. These programs may be run in real time and the relative geometry of the two items revealed in animated graphics. Several modes of running are provided for; full reports, messages, and abortion of further computation once a single interference has been detected, can be called for.

For testing Method I, the parallelepiped-arm has been given a 3D incremental spatial trajectory using the arbitrary parametric transformations $dx = at$, $dy = bt^2$, $dz = ct^3$, $\theta = et$, $\phi = ft$, $\varphi = gt$ (where t is a parameter or measure of time) for increments of displacement and angular rotation of axes about the (initially) lower, front, left-hand corner of the parallelepiped. This trajectory, whose shape is of no particular interest to the main theory of interference, takes the parallelepiped through a sequence of gyrations in the vicinity of a finial placed in an arbitrary position in space.

There is some collision at parts of the trajectory. If the paraxial 'boxes' enclosing both the finial and the parallelepiped-arm are determined (a standard operation) it may readily be ascertained whether they overlap, three dimensionally. If not, the program will conclude that no interference or collision can occur and will move on the next step and do so very rapidly, virtually instantaneously.

If the boxes overlap, the program proceeds to exami[illegible]tematically each and every vertex of the finial-workpiece to see if it lies within the volumetric space of the arm. If so, the sum of the tetrahedral volumes formed between the vertex and every triangular facet of the arm will add (within some very small selected error) to the known total enclosed volume of the arm. If not, the sum will be greater.

The three output options are as follows. Graphics of the workpiece and arm at successive intervals of time (with chosen dwell times for display), the workpiece being fixed in space, may be requested together with the printing of a cross at each workpiece vertex which interferes. In the course of progress, a 'cloud' of interfering vertices is printed, thus 'shading' regions of interference or collision.

The examination of all workpiece vertices may be curtailed if so few as one vertex is found to collide, thus reducing the time required for a complete examination of the whole trajectory. Full printed reports on collisions and signal 'bells' may be arranged if desired.

In many practical cases it is anticipated that the test of overlapping 'boxes' will indicate non-collision, virtually instantaneously; probably only one or two of the links of an articulated arm will collide at a few instants in a comprehensive complete robot arm trajectory. In this event there may be only one or two relocations of a link to deal with via the inverse kinematic problem. We have not dealt with this problem here.

For explaining and demonstrating Method II, two programs have been appended. The one in BASIC treats a specially selected relative location of workpiece and arm (Figure 18) designed to show how the general theory detects edges which are parallel to faces and thus intersect them only at infinity. Also vertices of one solid which lie in the planes of the other are identified. The program checks for edge-facet interference only one way; the test has not been reciprocated as has been done in the FORTRAN program to follow. Intersections of edges and faces within the bounds of each are detected and the coordinates of such points printed out.

The FORTRAN 77 version of the program for Method II has been framed to accept more complicated polyhedra than the most rudimentary pair of eight vertexed ones. It also processes a parametrically defined trajectory of the 'arm-link' and exchanges the data of link and workpiece at each instant examined to effect a reciprocal check. This program will detect all possible cases of collision by examining in excess of 400 possible cases of edge-facet relationship. In most cases the paraxial boxes will quickly indicate no possibility of collision and detailed analysis will be unnecessary and will be aborted.

Again, various options for output in graphical or printed alpha-numeric form are available.

Both FORTRAN programs are explained in more detail by remarks in the program listings.

Conclusions - Comparison of Methods I & II

Method I requires at least one only of the two items involved - the workpiece or the arm-link - to be densely specified by vertex coordinates. For a complex workpiece such a dense specification may be inherent in the problem; it may be essential to use many vertices to obtain an adequately detailed definition of that workpiece. When the above condition applies, reciprocation of the data and its reprocessing is not necessary with Method I as it always is with Method II.

The method depends on repeated evaluation of a 24 term determinant giving the volumes of numerous tetrahedra; hence it is based on the TETRAHEDRAL CONCEPT.

Method II works infallibly for any two polyhedra, even those with few vertices and thus of sparse specification. The method is based on standard relationships in analytical coordinate geometry. It requires the sparse data to be processed repeatedly in a series of algorithms and over 400 times for each instantaneous workpiece-arm relationship. However in many cases a first use of paraxial 'boxes' to test the feasibility of collision will prove negative and will permit abortion of detailed calculations.

To compare Method II with Method I for two polyhedra each of few vertices, one of these, usually a parallelepiped-arm, can be redundantly specified by including as vertices the intersections of a set of closely spaced planes parallel to the small ends with the long sides. Thus the arm becomes a densely specified polyhedron. It is now merely necessary to apply Method I in reverse, one way, by examining the vertices of the arm to see if they are within the volume of the sparsely defined workpiece. This device should be used for a fair comparison of the two methods in the case of two sparsely defined polyhedra.

Trials with the programs and particular trajectories chosen indicate that the 'boxing' test checks the whole trajectory virtually instantaneously over all those intervals where there is no possibility of collision. Significant time is required for a full check of all cases where overlapping boxes indicate the possibility of collision. There seems to be little to choose between the methods where such a full check is made. Like-wise, if computation is aborted as soon as the first point of collision or intersection is detected, the scan of the whole trajectory without graphics seems to be virtually instantaneous by both methods.

References

1. Automatic Interference Testing and Avoidance in Robotics by J.P. Duncan, Zhang Zhen-tao and A.O. Steeves. Paper in typescript.

2. Detection of Edge Interference in Robotics, by J.P. Duncan. Paper in typescript.

3. Mechanical Properties of Arbitrary Homogeneous Bodies by the Tetrahedral Concept, by J.P. Duncan. Paper in typescript.

4. Sculptured Surfaces in Engineering and Medicine, by J.P. Duncan and S.G. Mair, Cambridge University Press, 1983.

5. Trajectory planning for Redundant Manipulators in the Presence of Obstacles, Kircanski, M. and Vukobratovic, M., Institute Minailo Pupin, Beograd, Yugoslavia, Proc., 5th CISM, pp. 43-50, 1984.

6. The Development of the Fundamental Design Process for Mechanical Engineering - An Interim Report, Medland and Jones, Proc., IASTED International Conference, Computer Aided Design, Editor M.H. Hamza, Nice, France, June 1984, pp. 92-95, Acta Press.

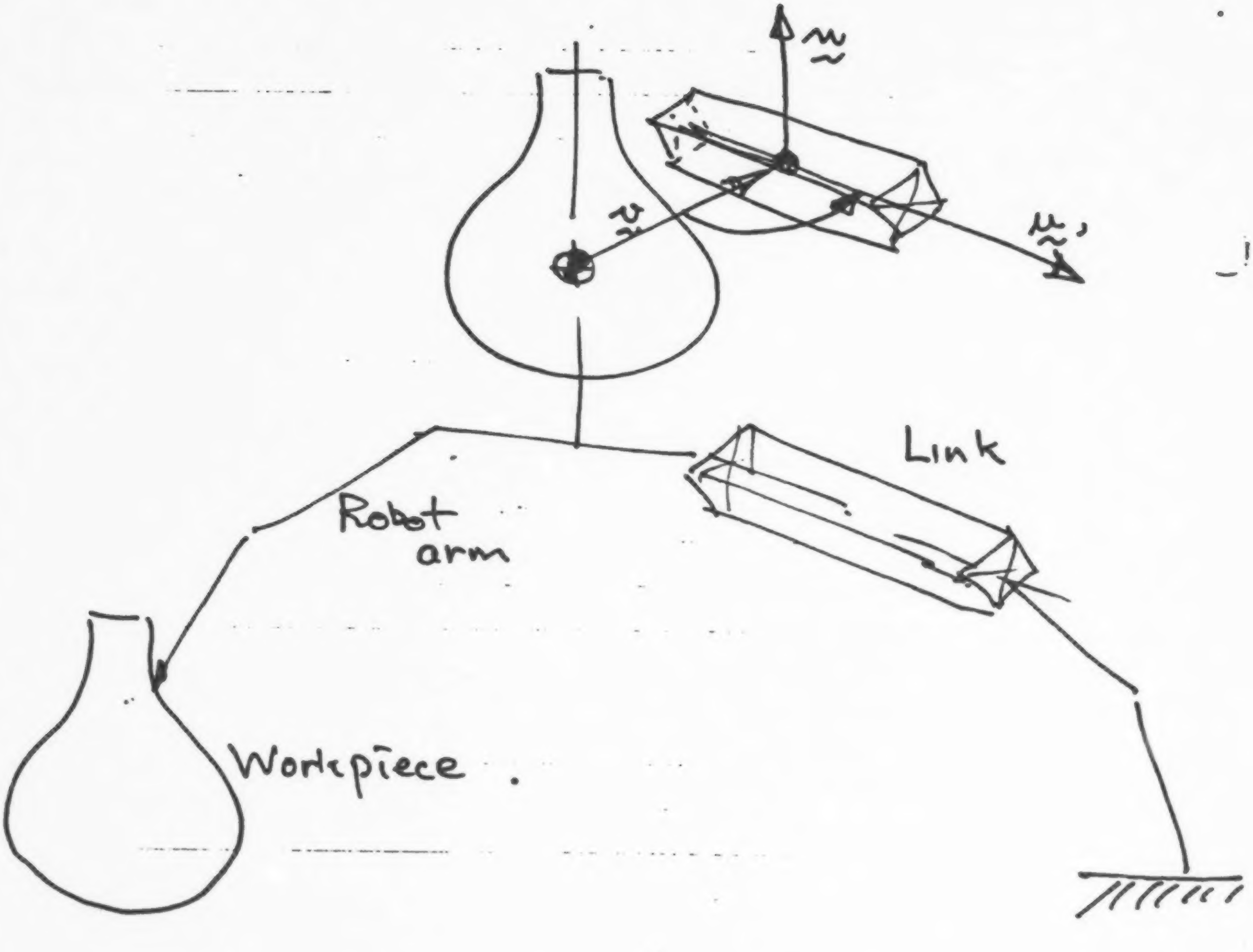
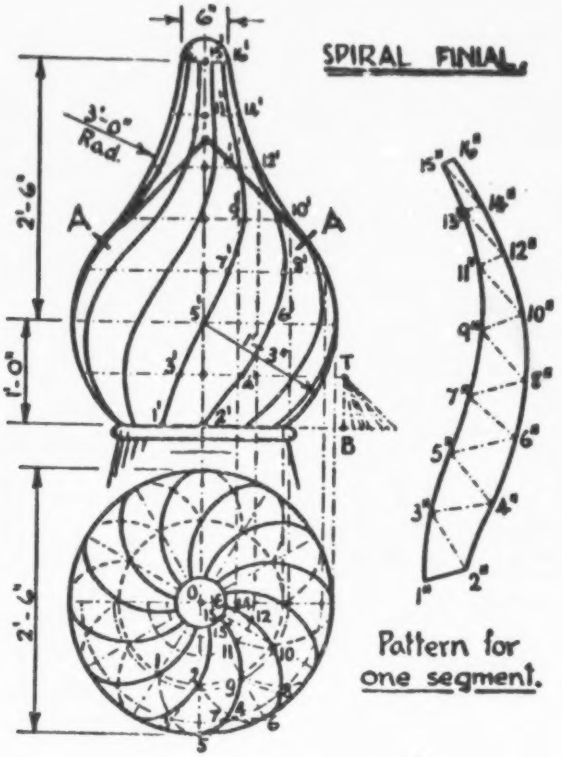
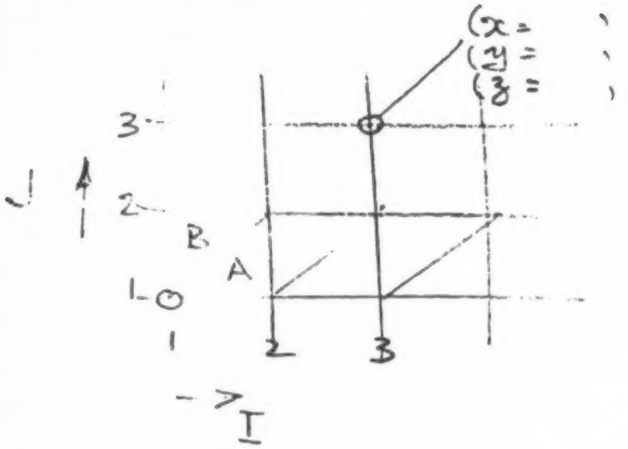

n
v
u'
Link
Robot arm
Workpiece

FIG 1


6"
3'-0" Rad.
2'-6"
1'-0"
2'-6"
A
A
T
B
15" 16"
14"
13"
12"
11"
10"
9"
8"
7"
6"
5"
4"
3"
2"
1"

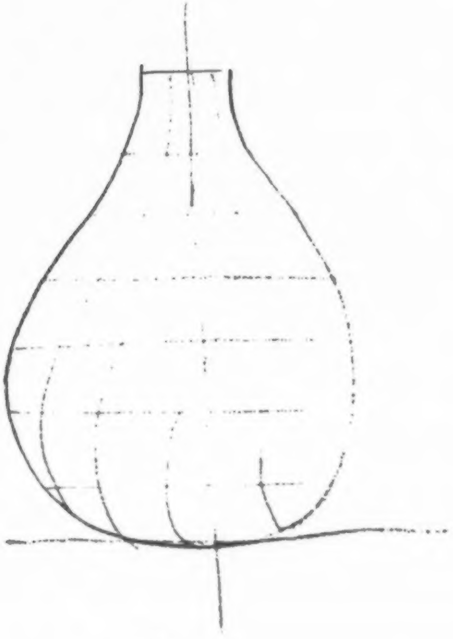
SPIRAL FINIAL.

Pattern for one segment.


(x =
(y =
(z =
3
2
1
B
A
O
J
I
1
2
3

"MERCATOR'S PROJ."

I,J INDICIAL GRID.



FINIAL

FIG 2.

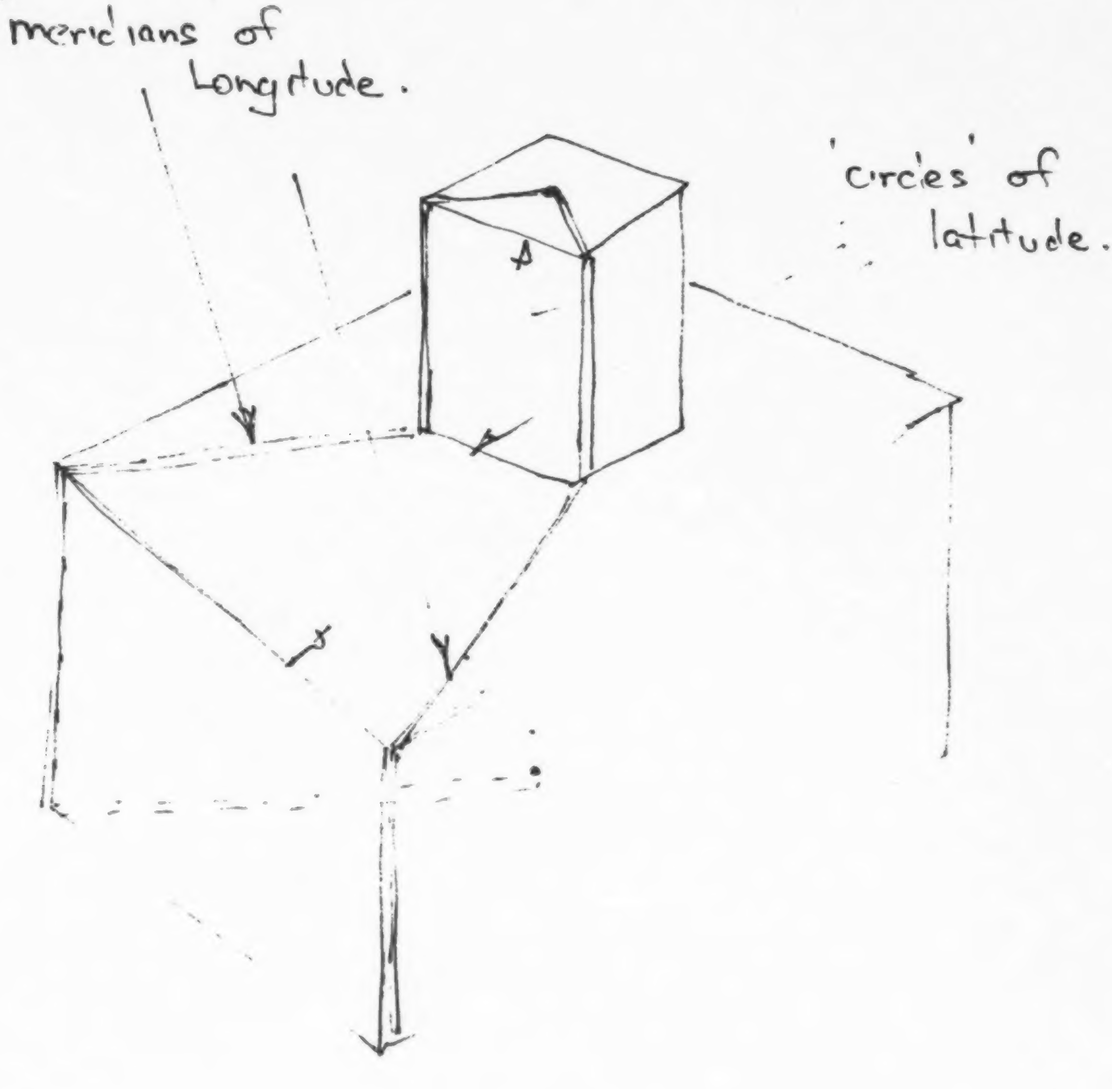
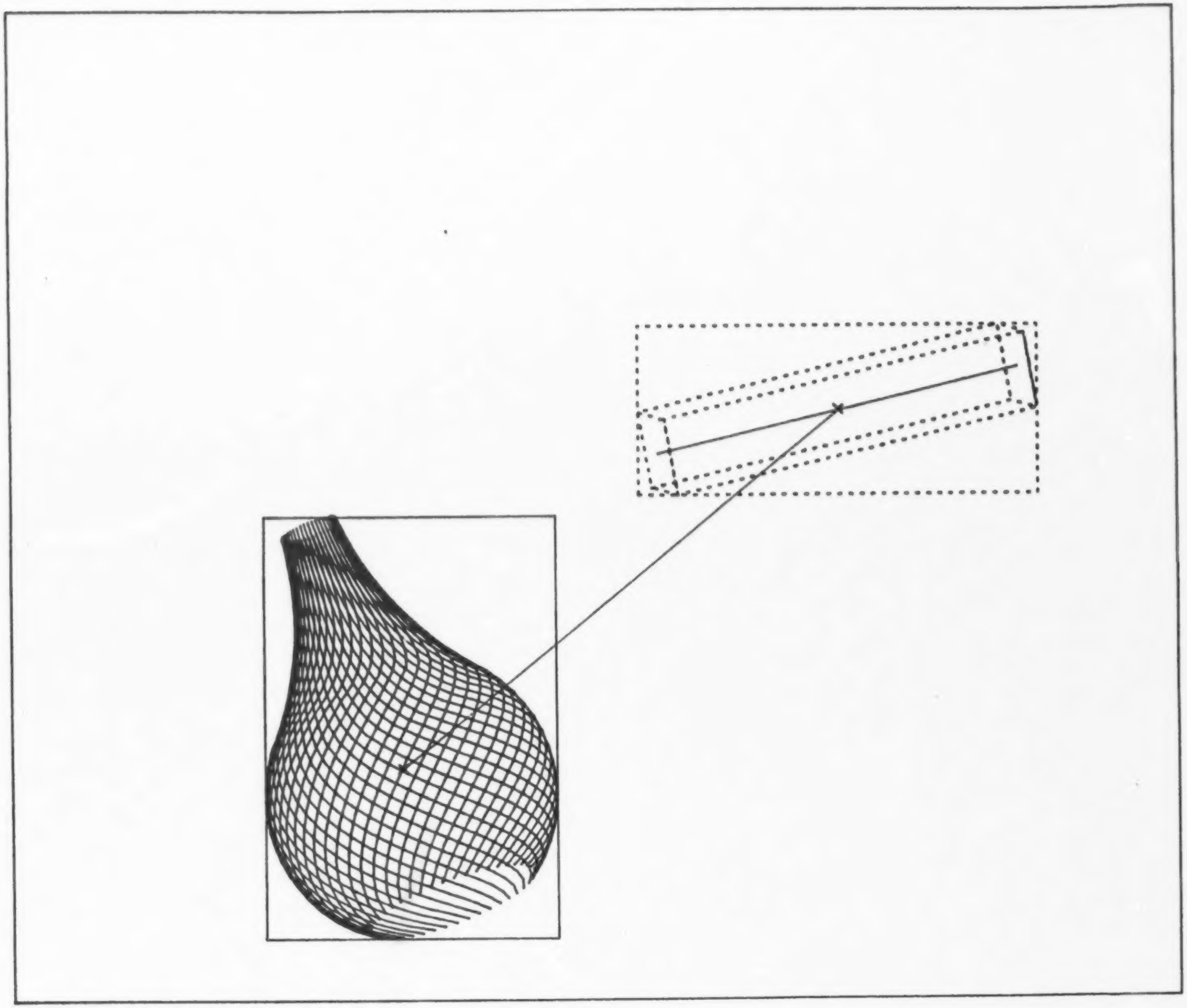
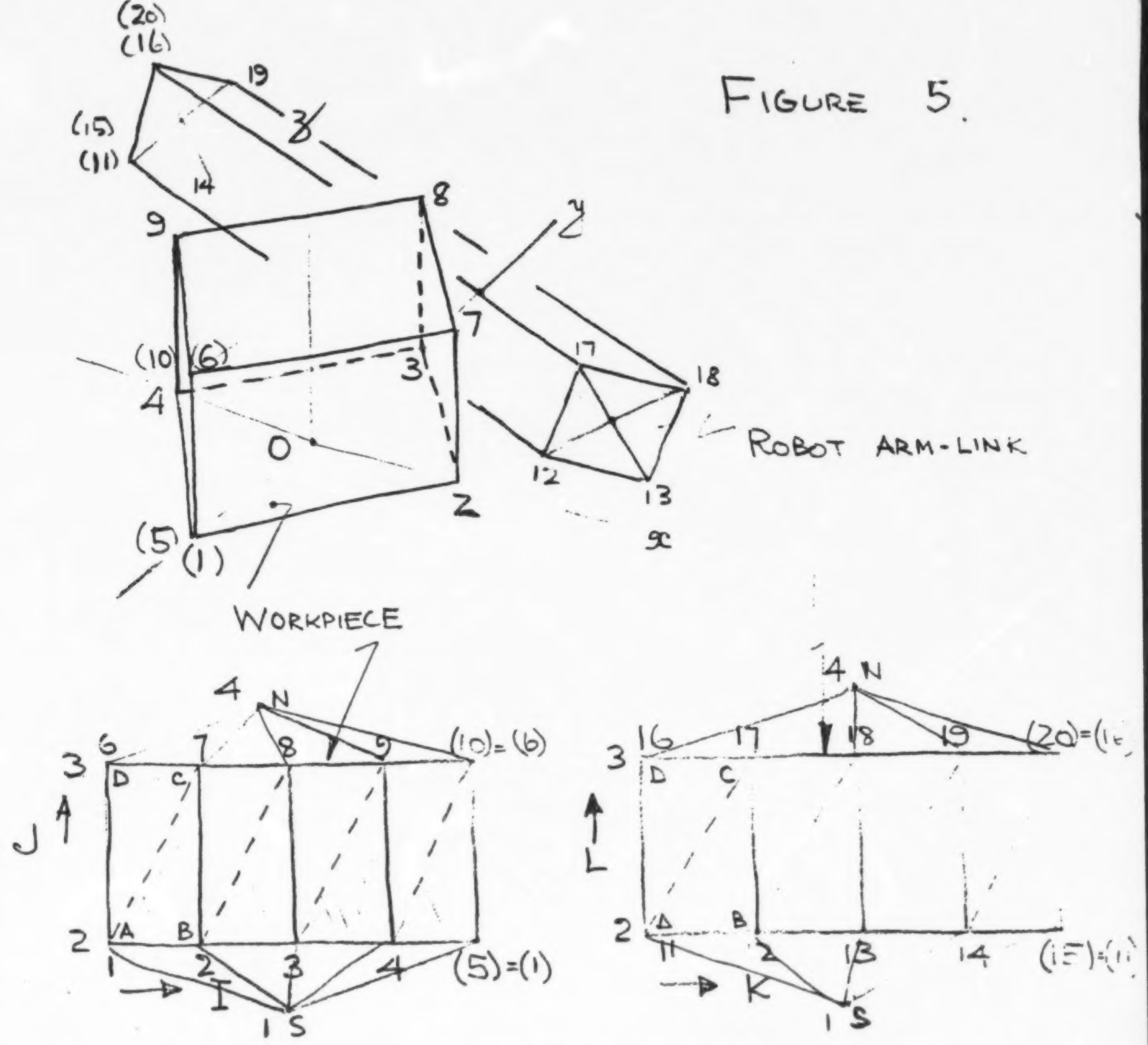

'meridians' of Longitude.
'circles' of latitude.

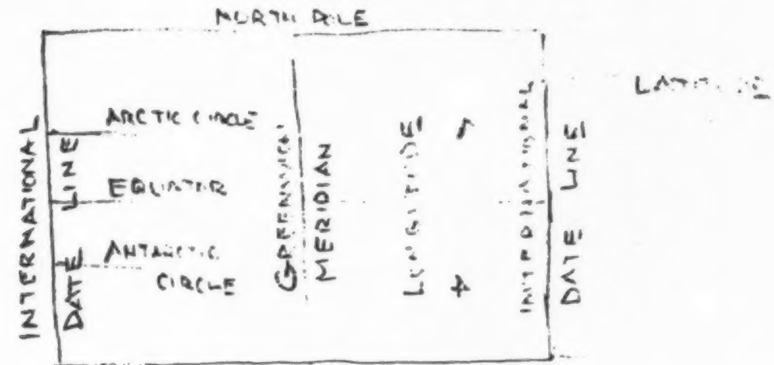
FIG 3.



INTERFERENCE TEST: NO INTERFERENCE AT ALL

FIGURE 5.


(20)
(16)
19
(15)
(11)
14
3
9
8
y
7
(10)
(6)
4
3
17
18
O
ROBOT ARM-LINK
12
13
2
x
(5)
(1)
WORKPIECE
4 N
3
6
7
8
9
(10)=(6)
D
C
J
2
A
B
1
2
3
4
(5)=(1)
I
1 S
4 N
16
17
18
19
(20)=(16)
3
D
C
L
2
A
B
11
12
13
14
(15)=(11)
K
1 S


NORTH POLE
LATITUDE
ARCTIC CIRCLE
EQUATOR
ANTARCTIC CIRCLE
INTERNATIONAL DATE LINE
GREENWICH MERIDIAN
LONGITUDE
INTERNATIONAL DATE LINE
SOUTH POLE

MERCATOR'S PROJECTION

2 THE WORLD IN HEMISPHERES-PHYSICAL & ASTRONOMICAL GEOGRAPHY 3

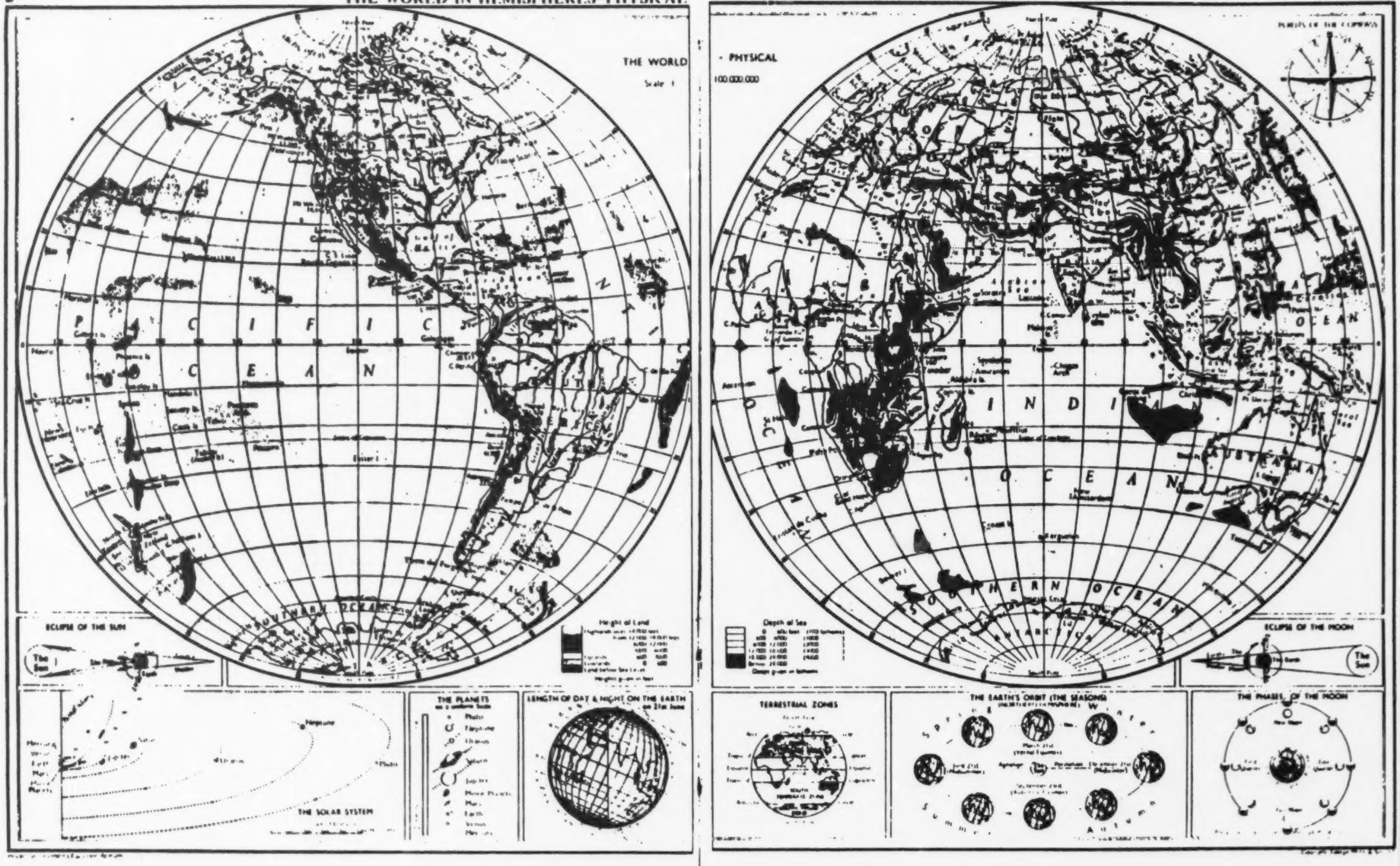
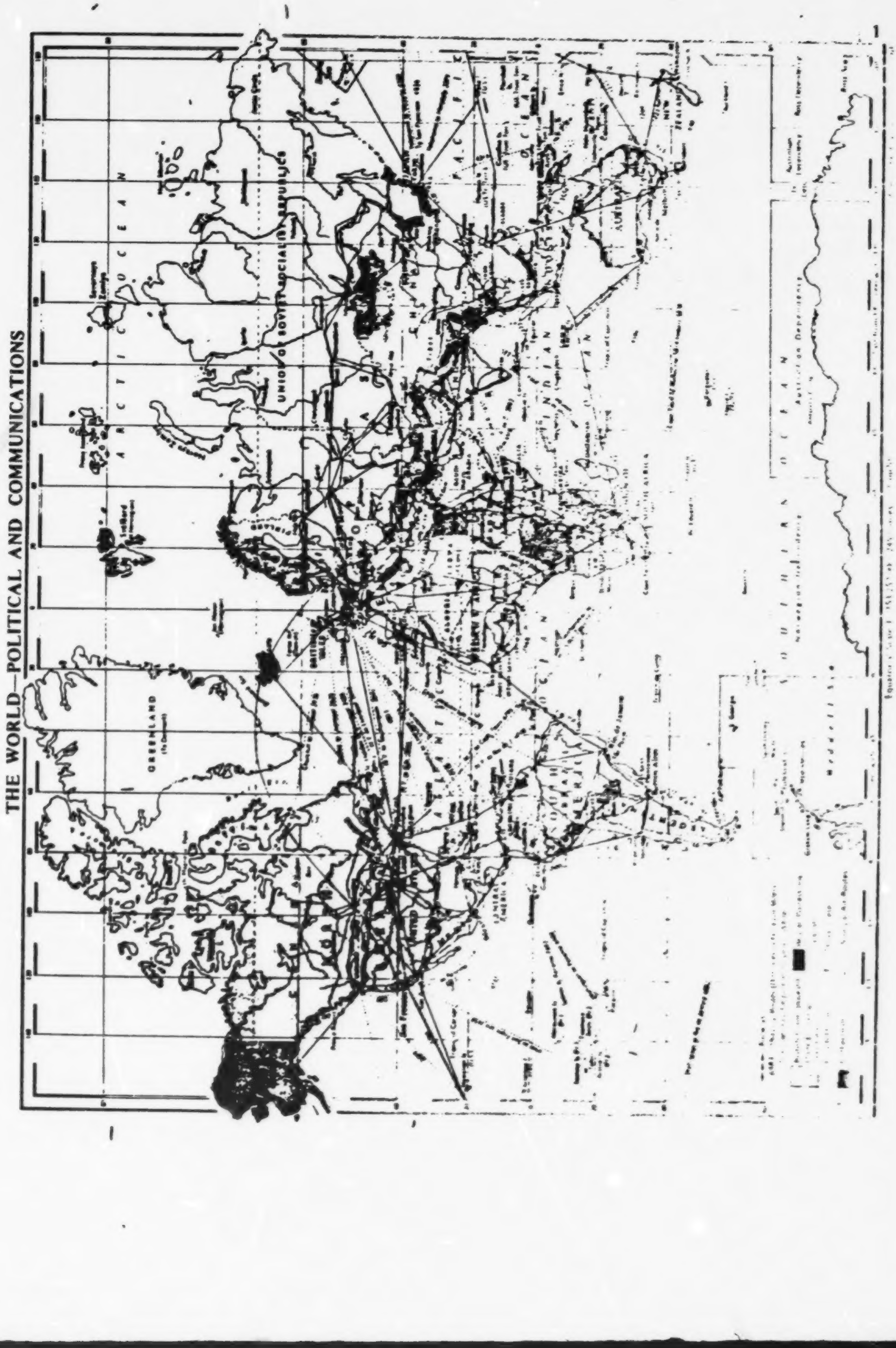

THE WORLD
Scale 1
PHYSICAL
100,000,000
POINTS OF THE COMPASS
PACIFIC OCEAN
INDIAN OCEAN
SOUTHERN OCEAN
AUSTRALIA
Height of Land
Depth of Sea
ECLIPSE OF THE SUN
The Sun
ECLIPSE OF THE MOON
The Sun
THE SOLAR SYSTEM
Neptune
Pluto
THE PLANETS
Pluto
Neptune
Uranus
Saturn
Jupiter
Mars
Earth
Venus
Mercury
LENGTH OF DAY & NIGHT ON THE EARTH on 21st June
TERRESTRIAL ZONES
THE EARTH'S ORBIT (THE SEASONS)
March 21st
THE PHASES OF THE MOON
New Moon

Fig. 6

161



THE WORLD—POLITICAL AND COMMUNICATIONS

GREENLAND
ARCTIC OCEAN
UNION OF SOVIET SOCIALIST REPUBLICS
INDIAN OCEAN
PACIFIC
ATLANTIC
AUSTRALIA
NEW ZEALAND

FIG 7.

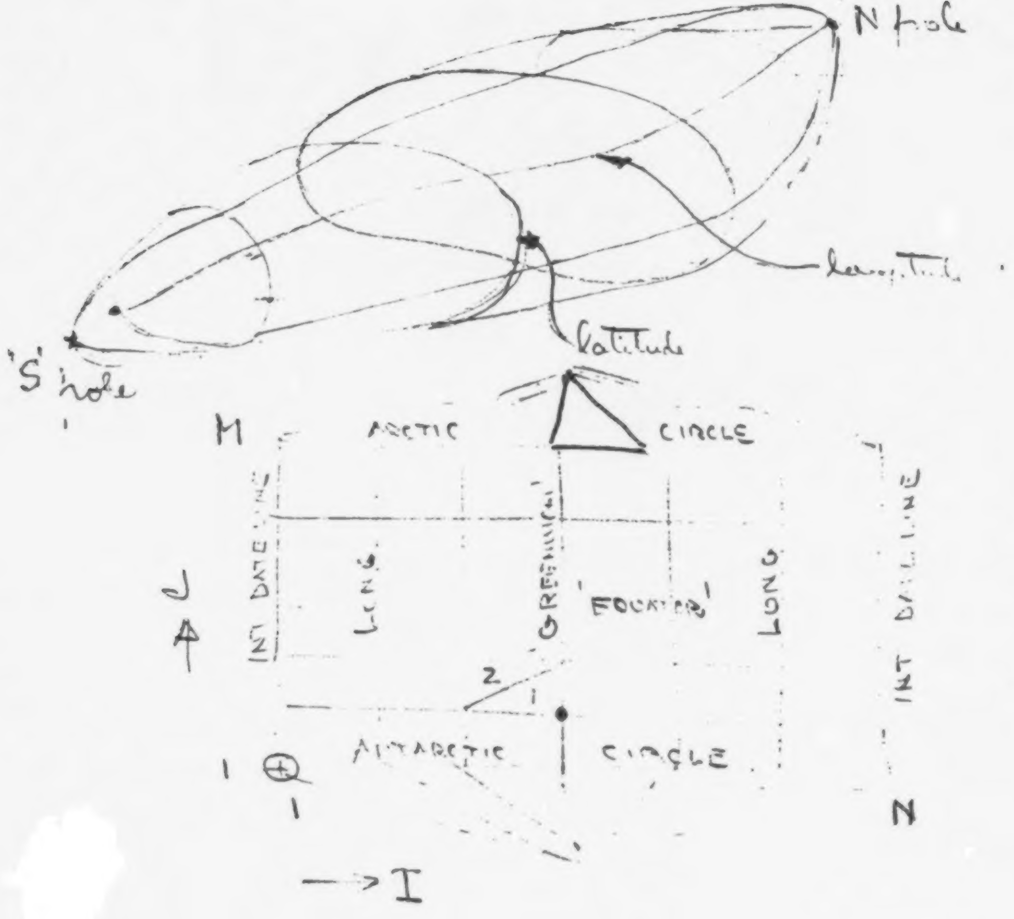
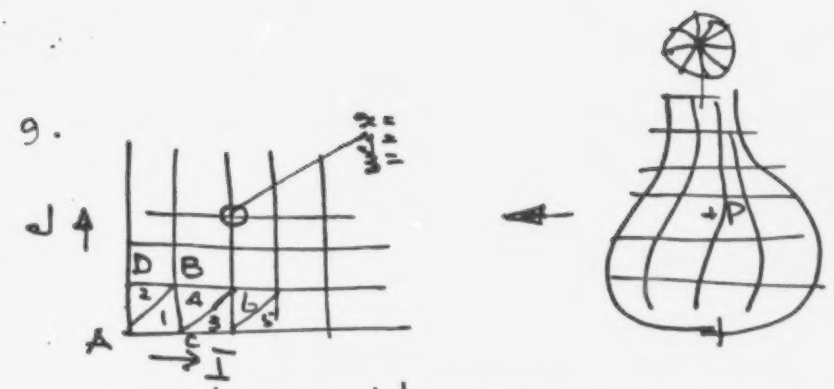

'N' pole
longitude
latitude
'S' pole
M
ARCTIC
CIRCLE
INT DATE LINE
LONG.
GREENWICH
'EQUATOR'
LONG.
INT DATE LINE
2
1
ANTARCTIC
CIRCLE
N
J
→ I

Fig 8

FIGURE 9.


x =
y =
z =
J
D
B
2
4
1
3
6
5
A
C
I
P

'MERCATOR'S' PROJECTION OF FINIAL.

FIGURE 10

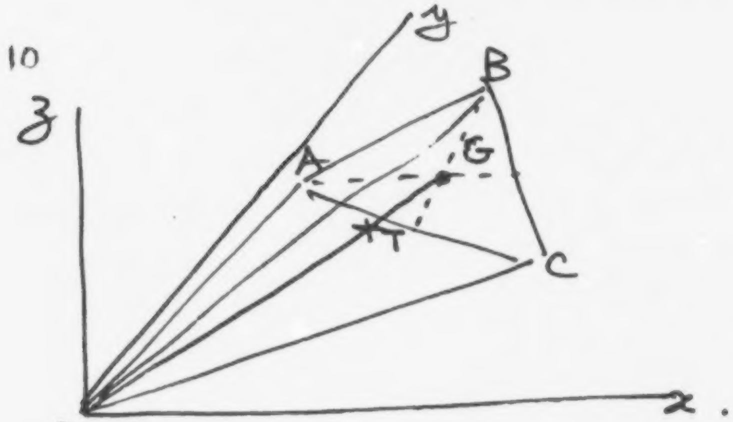
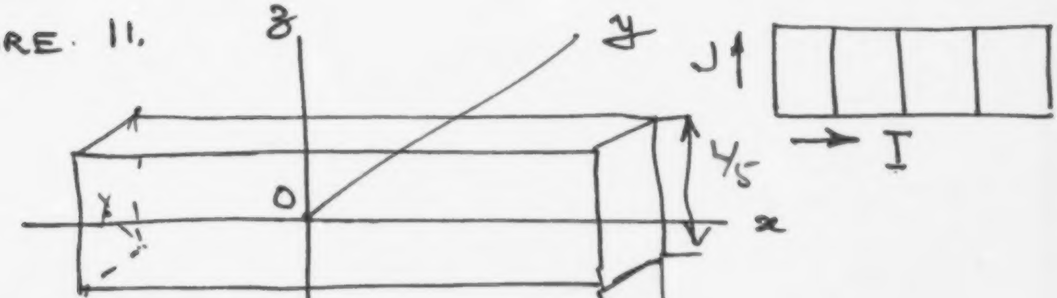
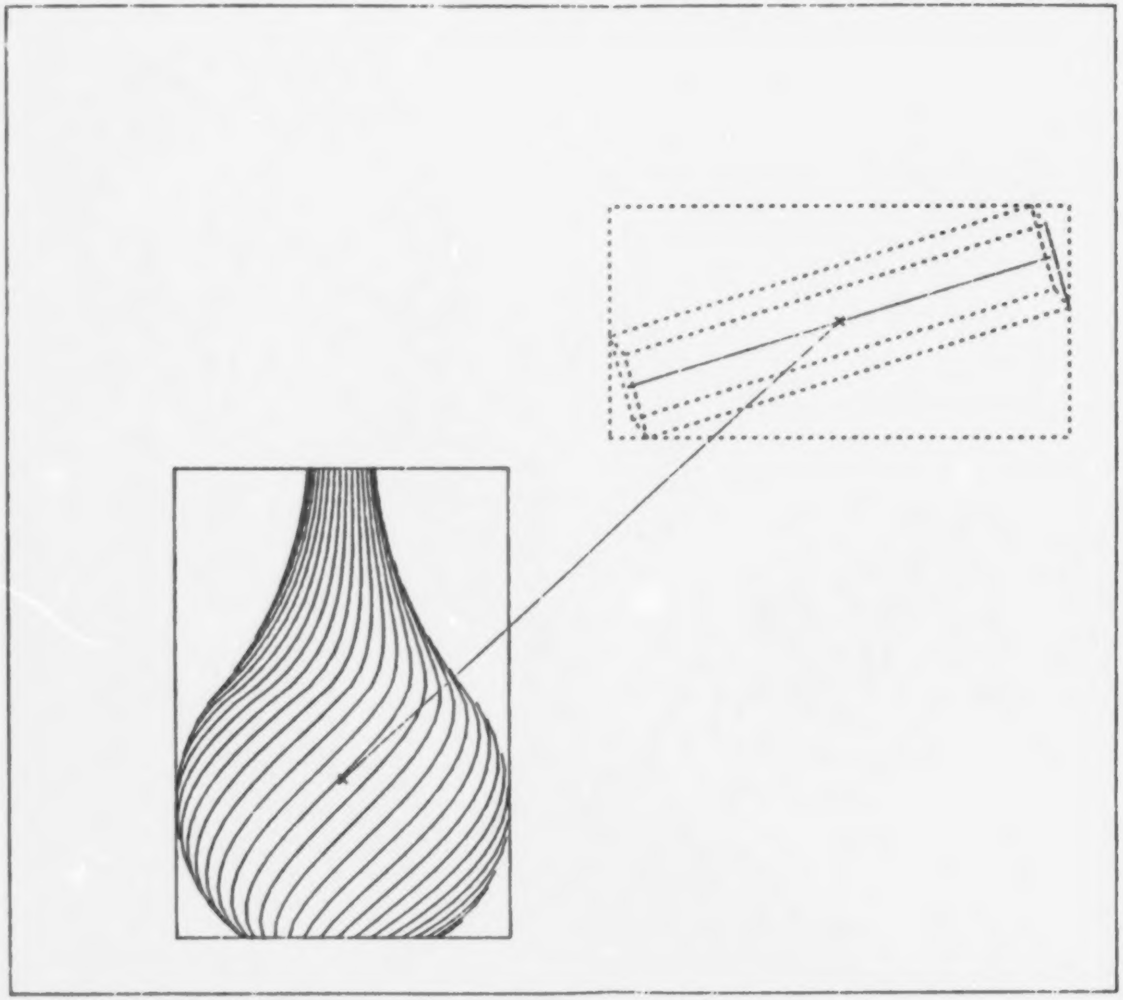

y
B
z
A
G
T
C
x.
D=O

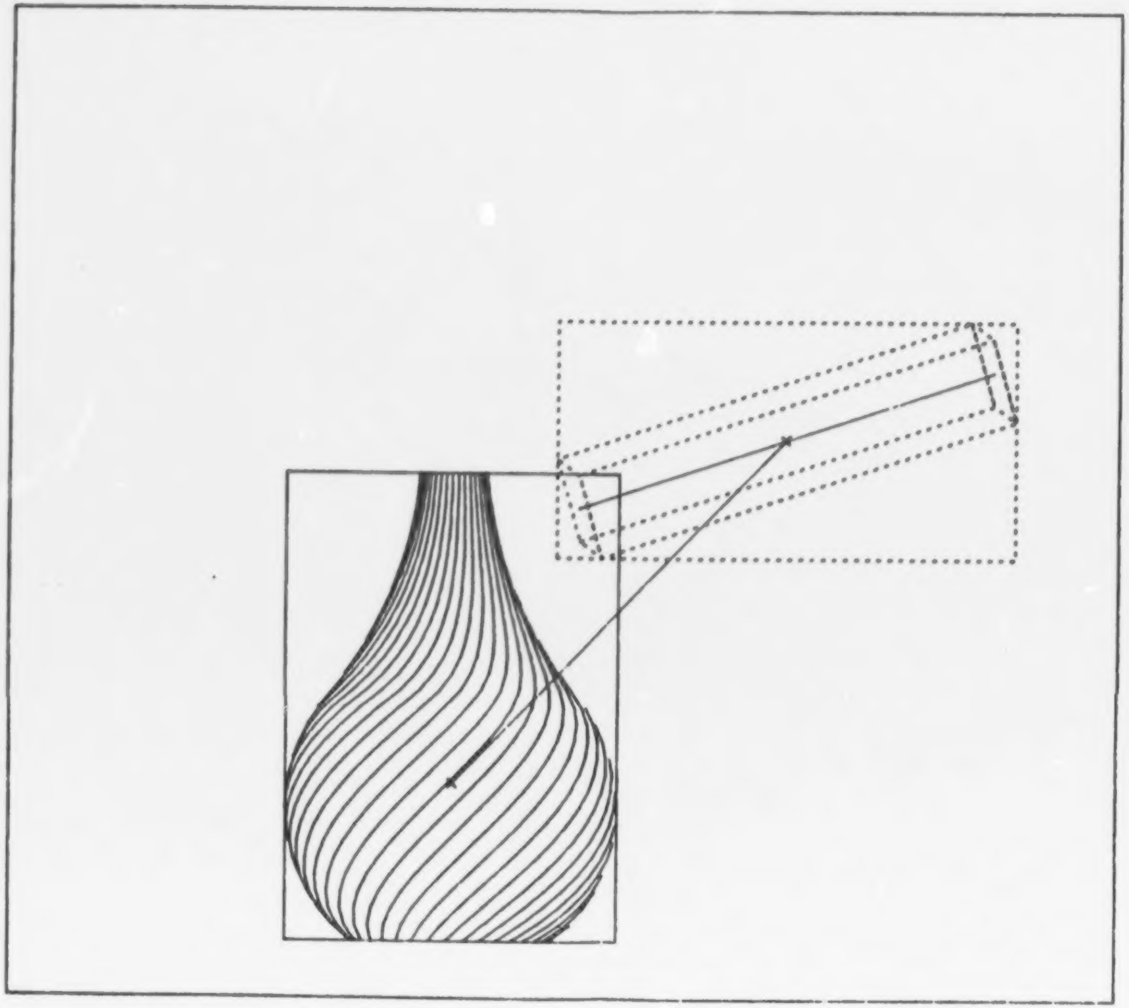
FIGURE 11.


z
y
J
I
4/5
O
x
L
4/5



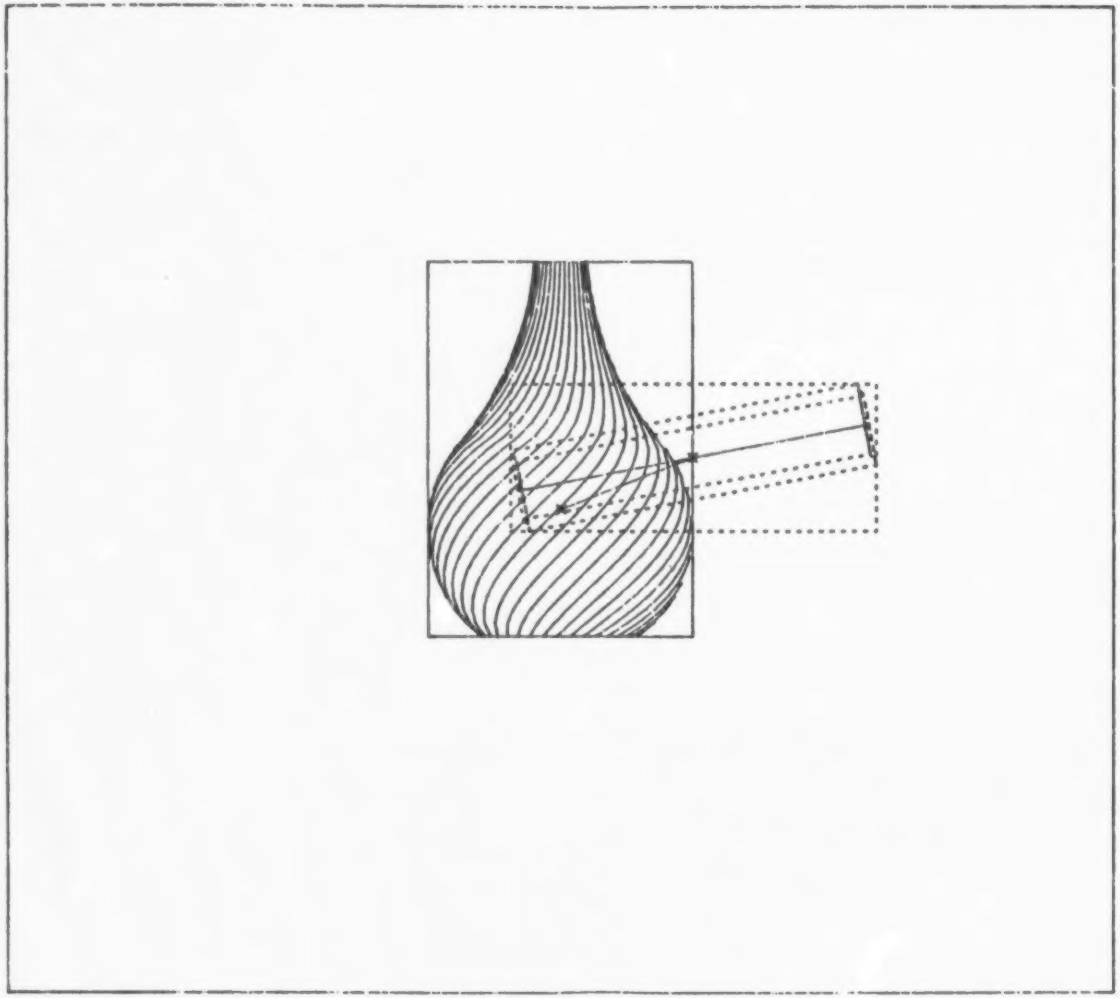
INTERFERENCE TEST

FIG 12



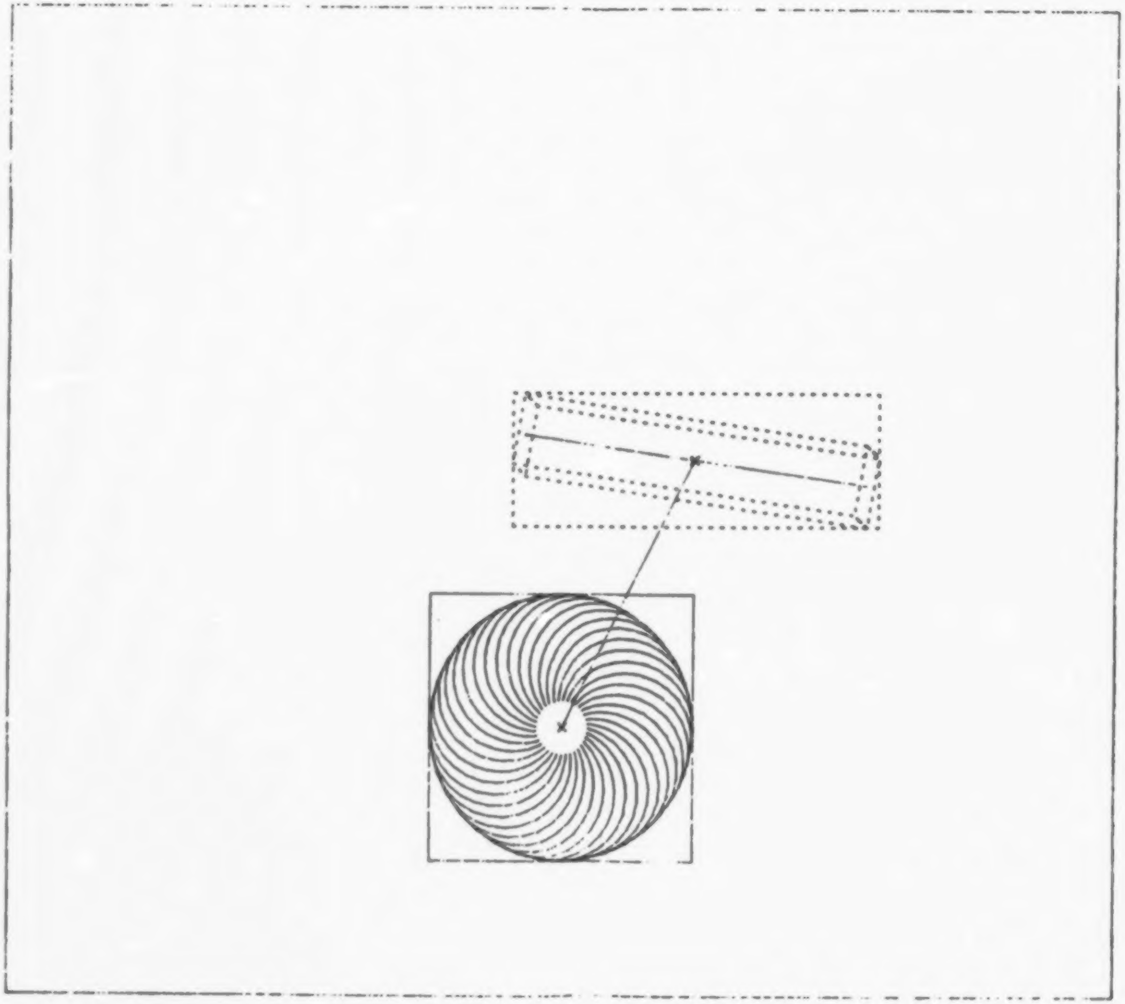
INTERFERENCE TEST

FIG13



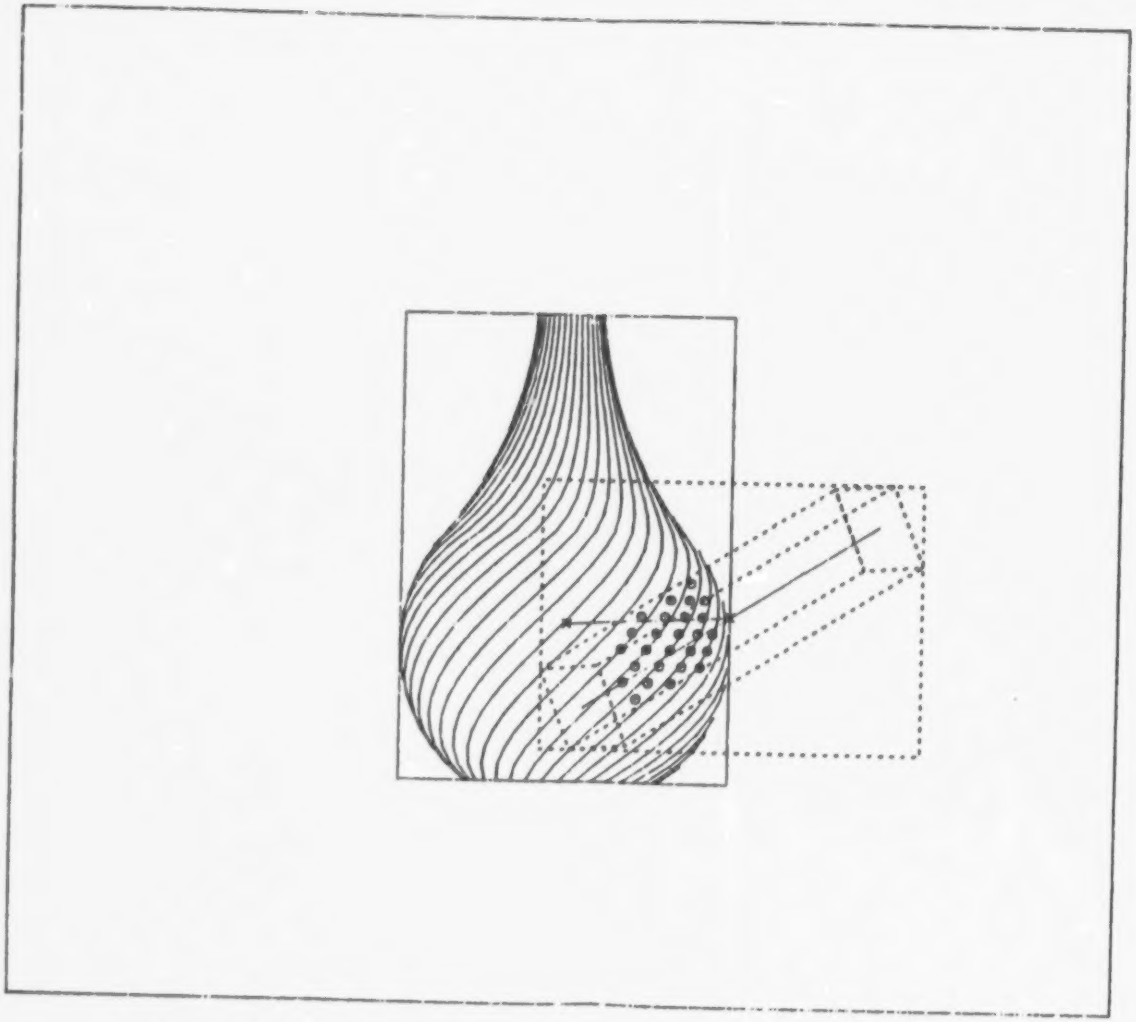
INTERFERENCE TEST

FIG 14.



INTERFERENCE TEST

FIG 15



INTERFERENCE TEST

FIG16

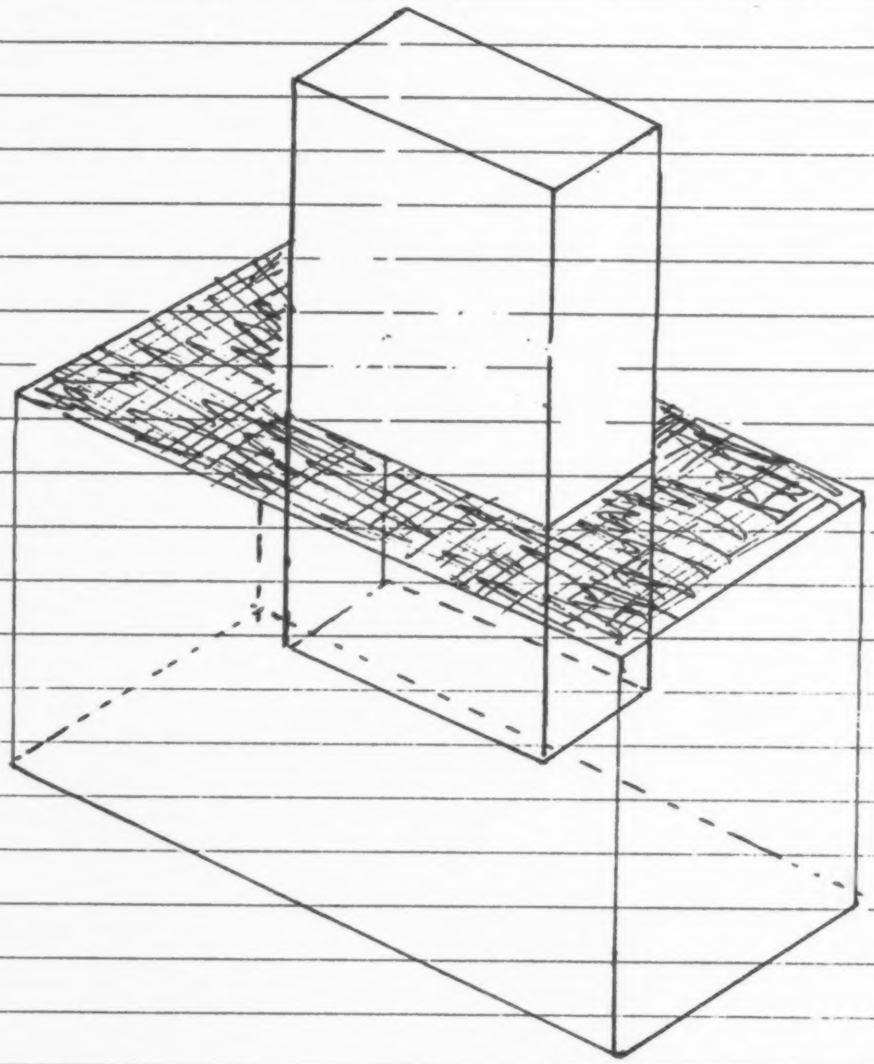


FIGURE 17

THE LARGE PARALLELEPIPED HAS NO EDGE WHICH PENETRATES ANY FACE OF THE SMALL ONE.

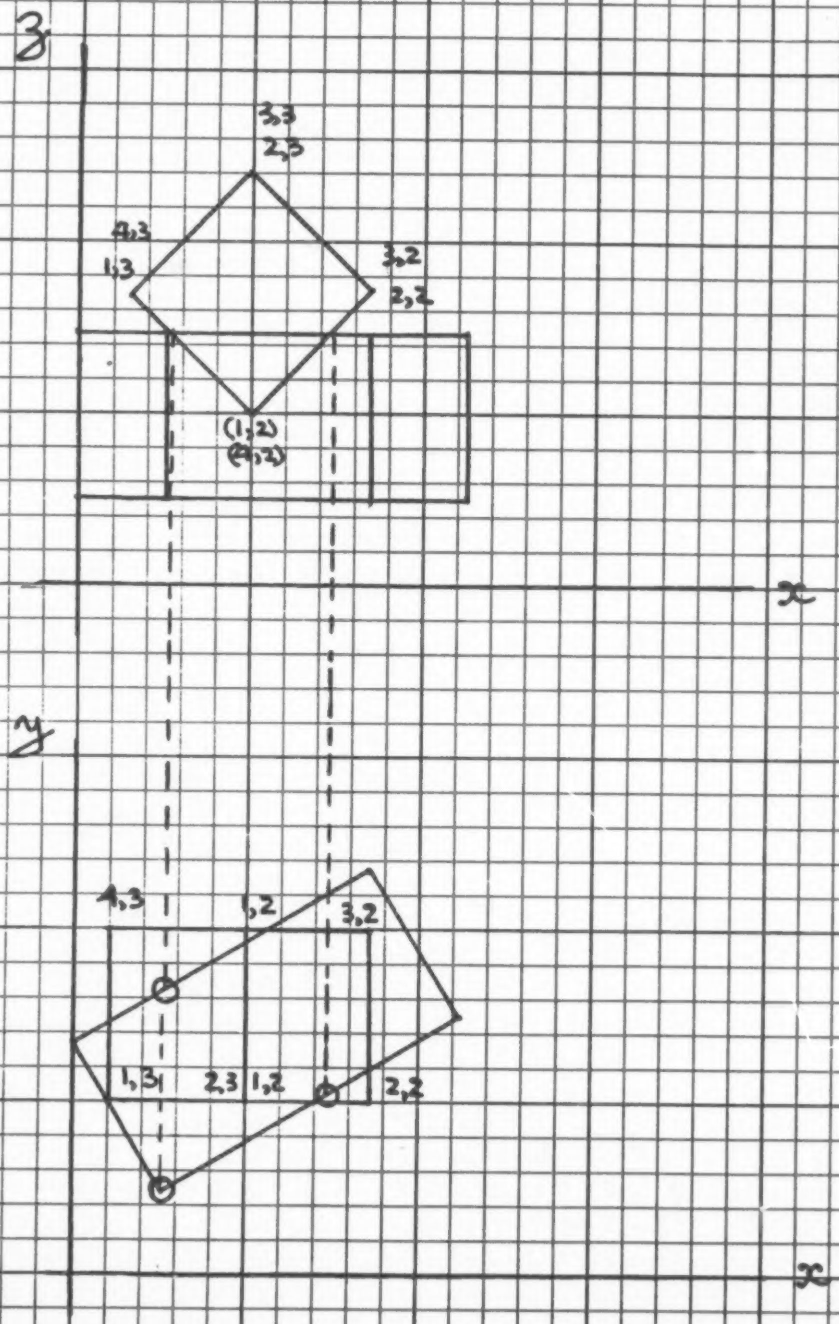

z
3,3
2,3
4,3
1,3
3,2
2,2
(1,2)
(4,2)
x
y
4,3
1,2
3,2
1,3
2,3
1,2
2,2
x

FIGURE 18

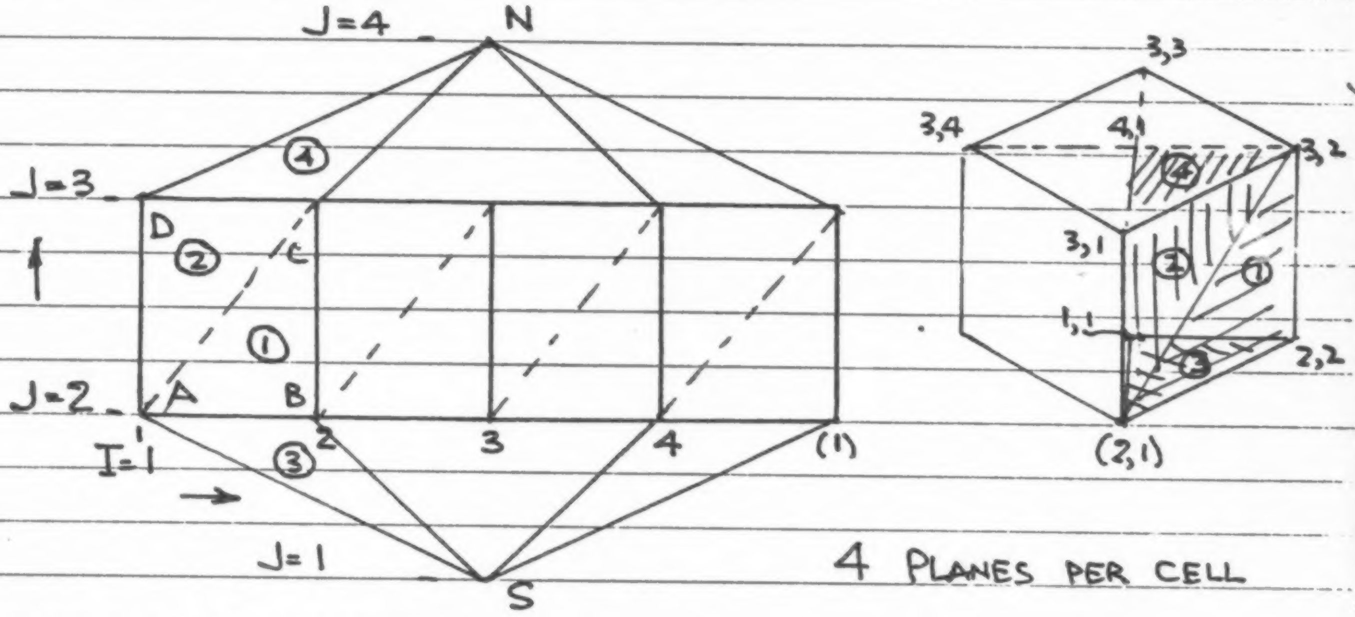

J=4
N
J=3
D
②
C
④
①
J=2
A
B
I=1
1
2
3
4
(1)
③
J=1
S
3,3
3,4
4,1
3,2
④
3,1
②
①
1,1
③
2,2
(2,1)
4 PLANES PER CELL

FIGURE 19

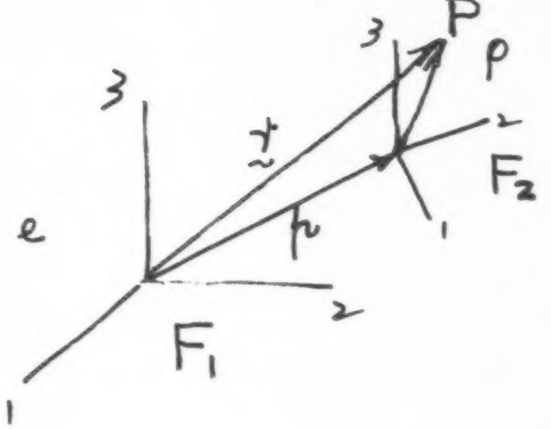


FIG. 20

Transformation of displacement with rotation of axes.
Homogeneous transformation.

We will require to "place" the parallelepiped (robot arm-link) at various positions in the global field. To do this, use homogeneous transformation as follows (e = unit-vector)

$$\underset{\sim}{r} = \begin{bmatrix} e_{11} & e_{21} & e_{31} & p_1 \\ e_{12} & e_{22} & e_{32} & p_2 \\ e_{13} & e_{23} & e_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ 1 \end{bmatrix}$$

$$\underset{\sim}{r} = H_{1,2} \cdot \rho$$

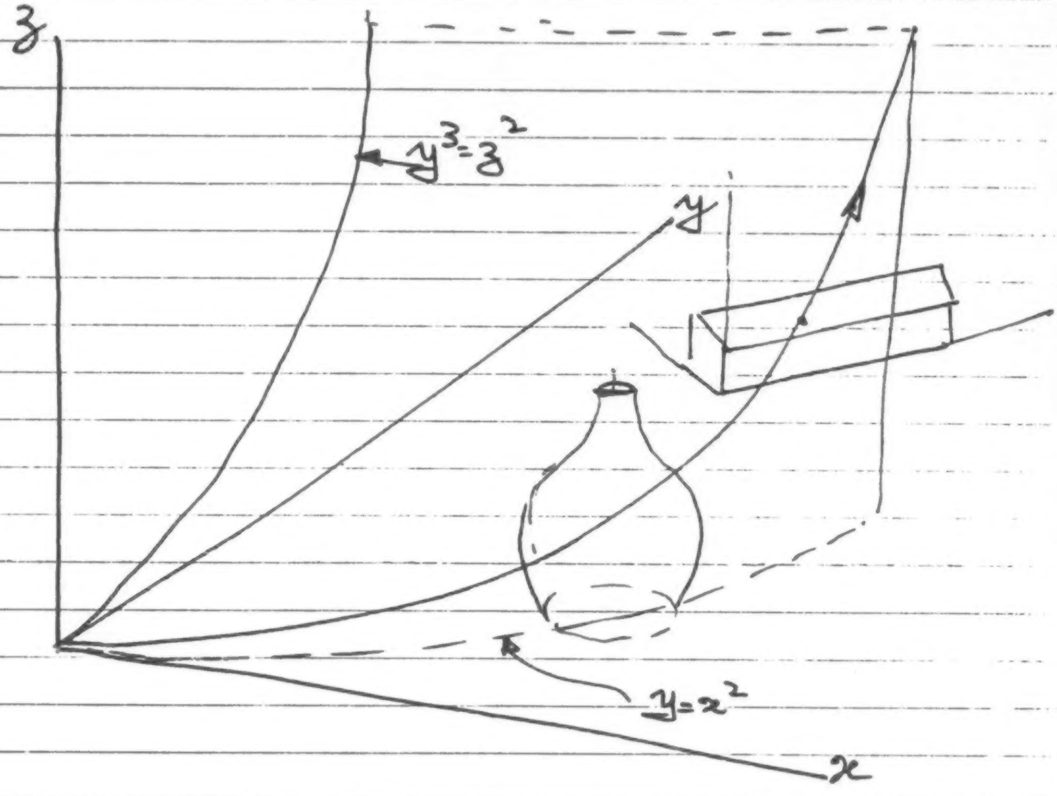
This gives P in F_2 projected onto F_1

The components of unit-vector are the direction cosines w.r.t. to basic vector-axes

Each link(s) of parallelepiped needs to be transformed

FIG. 21

Curve used to create an example — centroid of parallelepiped follows curve. Axes rotate as function of t. Vial remains stationary so that arm will interfere at some range of t.



$x = at \quad y = bt^2 \quad z = ct^3$

t = parameter

Rotations as linear function t

$\theta = et$
$\phi = ft$
$\gamma = gt.$

This is the situation which can be viewed on terminal screen.

APPENDIX III - SOFTWARE DOCUMENTATION OF 'AUTOWELD'

III.1 Functional Description of 'Autoweld'

AUTOWELD is an interactive graphics program containing a graphics loop, which updates the screen with each pass. With the exception of the routine 'intest', which shows the robot and workpiece while a welding trajectory is being sought, there are no other graphics loops. This eliminates repetition of the graphics overhead and allows simple interaction.

The keyboard is polled once each pass, and if user input is detected, appropriate flags are set. These flags define modes and are used to enable and disable various functions. The key strokes are queued, so they cannot be missed.

The user can interactively change graphical viewing parameters using the mouse. Certain workstation parameters such as joint positions can also be interactively varied.

When the display is static, all unnecessary processing is bypassed in the loop. This minimizes the response time to user input. Any processing done within the loop increases the response time.

In order to avoid introducing other graphical loops, some subroutines cannot be nested in a natural way. Rather than incorporating a loop into a lower level subroutine, the subroutine executes just one pass each time it is called. All relevant parameters are locally saved by the subroutine so that each time it is called it performs the next loop. This method allows interactive user control at all times, by effectively executing the high-level graphics loop within the lower-level subroutine loop.

III.1.1 Subroutines Grouped by Function. A number of basic functions can be identified within AUTOWELD. The subroutines can be grouped according to these functions. Subroutines which are difficult to classify may appear in several groups.

1. Initialization

   init initgr rdmodl store

   Initialization is performed just once, when AUTOWELD is started.

2. Interference detection

   finbes finsep plist

   These subroutines are also used for the first part of the hidden surface removal algorithm.

3. Mathematical Functions

   adjcnt cdist cvhull disjnt edges edgvrt etest
   ftest getsep pface splsid

   These perform detailed calculations and low level processing. These calculations are used in interference detection and hidden surface removal.

4. Hidden Surface Removal

   finbes finsep plist psort obsort obswap ohsurf ohtest
   osort2

   Show a shaded image with hidden surfaces removed if desired.

5. Graphical depiction

displl drpart drpoly drscrn drseam drsequ drtext

drwire drwkst setmen

Generate graphical output.

6. Interactive control

addpos dfpath evpick input mkseam mksequ movpos

picksm rempos revdir rmseam smpick unweld

Interact with the user to control AUTOWELD.

7. Expert welder

calcs exprt1 exprt2 wgeom wparam

Simple welding routines incorporated for demonstration purposes.

8. Kinematics

inpath intest joints limits mtable noap5 tran

trandb transp

Perform kinematic calculations and transform the data base.

9. Weld trajectory generation

config genpat intest joints limits tolrot worpos

Iteratively search for a feasible welding trajectory.

III.2 'AUTOWELD' USERS GUIDE

AUTOWELD is not a finished commercial package, but rather is a research tool which demonstrates the feasibility of interactive off-line programming and provides a basis for further development.

This guide is intended to instruct a user in the operation of the current version of AUTOWELD and is subject to change with future versions.

III.2.1 Compiling AUTOWELD. Use the UNIX utility 'make'. The complete specifications for AUTOWELD are contained in file 'Makefile'. AUTOWELD must be recompiled each time a change is made to the source code. To recompile, enter 'make'.

III.2.2 Input Files. The following input files must exist in the correct format:

CAD.DAT-	CAD seam reference data
CADLST.DAT-	CAD seam pointers
GEOM.DAT-	geometric model of the workstation
SETUP.DAT-	parts to be tested and shown
WKSTAT.DAT-	fixed kinematic data about the workstation
WPCLST.DAT-	CAD welding parameters

If a previous state is to be restored, then the following files must exist and contain valid data:

ITABLE.DAT-	pointers to positions on weld trajectory
ILOCAT.DAT-	positions on weld trajectory
PTABLE.DAT-	pointers to inter-weld positions
PLOCAT.DAT-	inter-weld positions
SAVE.DAT-	previous AUTOWELD state. Contains pointers to seam and weld data.

The following files can be defined by the user:

1. GEOM.DAT

GEOM.DAT contains all topological and metric information about objects in the workstation. This information comprises a boundary representation geometric model. Additional information is contained about known separating planes. GEOM.DAT is generated by an external CAD system or by a geometric modeller. A rudimentary geometric modeller, GEM, has been developed to simplify user definition of different workpieces for testing and demonstration. GEOM.DAT can also be generated or modified manually using the editor.

2. WKSTAT.DAT

WKSTAT.DAT contains all information describing the workstation kinematics. The user can enter and modify this information using the editor. WKSTAT.DAT need only be changed when a different robot or positioning table is required.

3. SETUP.DAT

SETUP.DAT specifies which parts are to be tested for interference, and which parts are to be shown on the screen.

III.2.3 Detailed Contents of User-defined Files. Samples of the actual file contents are reproduced below. The lines of text also appear in the files as a guide to the user who is checking or modifying the entries. {Additional comments appear inside braces like this.} It is ESSENTIAL that the files correspond record for record with these samples. The input format of each record is list directed.

III.2.3.1 File GEOM.DAT

```
clusters CLUSTR
no. of clusters
clust. ID, first part, no.parts, first face, no. faces, first vert, no.
verts
4
1   1   1   1    6    1    8
2   2   1   7    6    9    8
3   3   2   13   12   17   16
4   5   1   25   6    33   8
PARTS
no. of parts
part ID,first face,no, of faces
5
1   1    6
2   7    6
3   13   6
4   19   6
5   25   6
known separating planes CSPLAN
no. of separating planes
index,part ID,other part ID,separating face ID
3
1   2   1   1
2   3   2   3
```

```
3   4   3   6

separating planes

index,vector,normal

1    +0.000000E+00    +4.000000E+01    +0.000000E+00    +0.000000E+00

+1.000000    +0.000000E+00

3    +0.000000E+00    +0.000000E+01    +0.000000E+00    +0.000000E+00

+0.000000E+00    -1.000000

6    +0.000000E+00    +0.000000E+01    +9.800001E+01    +0.000000E+00

+0.000000E+00    -1.000000

number of faces NFACE

30

vertex list VERTS

number of vertex pointers

face ID,no. of vertices, vertex IDs

120

1    4    1    2    6    5

2    4    2    3    7    6

3    4    3    4    8    7

4    4    4    1    5    8

5    4    1    4    3    2

6    4    5    6    7    8

7    4    9    10    14    13

8    4    10    11    15    14

9    4    11    12    16    15

10    4    12    9    13    16

11    4    12    9    13    16
```

```
12    4    13    14    15    16
13    4    17    18    22    21
14    4    18    19    23    22
15    4    19    20    24    23
16    4    20    17    21    24
17    4    17    20    19    18
18    4    21    22    23    24
19    4    25    26    30    29
20    4    26    27    31    30
21    4    27    28    32    31
22    4    28    25    29    32
23    4    25    28    27    26
24    4    29    30    31    32
25    4    33    34    38    37
26    4    34    35    39    38
27    4    35    36    40    39
28    4    36    33    37    40
29    4    33    36    35    34
30    4    37    38    39    40
vertex coordinates VCOORD
no. of vertices
vertex ID,coordinates X,Y,Z
40
1    -4.000000E+01    -8.000000E+01    +5.000000E+01
2    +3.600000E+01    -8.000000E+01    +5.000000E+01
3    +3.600000E+01    +8.000000E+01    +5.000000E+01
```

```
4   -4.000000E+01   +8.000000E+01   +5.000000E+01
5   -4.000000E+01   -8.000000E+01   +1.300000E+02
6   +3.600000E+02   -8.000000E+01   +1.300000E+02
7   +3.600000E+02   +8.000000E+01   +1.300000E+02
8   -4.000000E+01   +8.000000E+01   +1.300000E+02
9   -4.000000E+01   -4.000000E+01   +0.000000E+00
10   +4.000000E+01   -4.000000E+01   +0.000000E+00
11   +4.000000E+01   +4.000000E+01   +0.000000E+00
12   -4.000000E+01   +4.000000E+01   +0.000000E+00
13   -4.000000E+01   -4.000000E+01   +4.300000E+02
14   +4.000000E+01   -4.000000E+01   +4.300000E+02
15   +4.000000E+01   +4.000000E+01   +4.300000E+02
16   -4.000000E+01   +4.000000E+01   +4.300000E+02
17   -5.000000   -5.000000   +0.000000E+00
18   +5.000000   -5.000000   +0.000000E+00
19   +5.000000   +5.000000   +0.000000E+00
20   -5.000000   +5.000000   +0.000000E+00
21   -5.000000   -5.000000   +9.800001E+01
22   +5.000000   -5.000000   +9.800001E+01
23   +5.000000   +5.000000   +9.800001E+01
24   -5.000000   +5.000000   +9.800001E+01
25   +0.000000E+00   -5.000000   +9.800001E+01
26   +5.656800E+01   -5.000000   +1.545680E+02
27   +5.656800E+01   +5.000000   +1.545680E+02
28   +0.000000E+00   +5.000000   +9.800001E+01
29   -2.828400   -5.000000   +1.008284E+02
```

```
30   +5.373960E+01   -5.000000   +1.573964E+02

31   +5.373960E+01   +5.000000   +1.573964E+02

32   -2.828400   +5.000000   +1.008284E+02

33   -1.750000E+02   -1.750000E+02   -2.500000E+01

34   +1.750000E+02   -1.750000E+02   -2.500000E+01

35   +1.750000E+02   +1.750000E+02   -2.500000E+01

36   -1.750000E+02   +1.750000E+02   -2.500000E+01

37   -1.750000E+02   -1.750000E+02   +0.000000E+01

38   +1.750000E+02   -1.750000E+02   +0.000000E+01

39   +1.750000E+02   +1.750000E+02   +0.000000E+01

40   -1.750000E+02   +1.750000E+02   +0.000000E+01

faces adjacent to vertices,FACES2

no. of adjacent face pointers

vertex ID,no. of adjacent faces,face IDs

120

1   3   1   4   5

2   3   1   2   5

3   3   2   3   5

4   3   3   4   5

5   3   1   4   6

6   3   1   2   3

7   3   2   3   6

8   3   3   4   6

9   3   7   10   11

10   3   7   8   11

11   3   8   9   11
```

12	3	9	10	11
13	3	7	10	12
14	3	7	8	9
15	3	8	9	12
16	3	9	10	12
17	3	13	16	17
18	3	13	14	17
19	3	14	15	17
20	3	15	16	17
21	3	13	16	18
22	3	13	14	15
23	3	14	15	18
24	3	15	16	18
25	3	19	22	23
26	3	19	20	23
27	3	20	21	23
28	3	21	22	23
29	3	19	22	24
30	3	19	20	21
31	3	20	21	24
32	3	21	22	24
33	3	25	28	29
34	3	25	26	29
35	3	26	27	29
36	3	27	28	29
37	3	25	28	30

```
38   3   25   26   27
39   3   26   27   30
40   3   27   28   30
faces adjacent to faces,FACES3
no. of face pointers
face id,no. of adjacent faces,face IDs
120
1    4   2    4    5    6
2    4   1    3    5    6
3    4   2    4    5    6
4    4   1    3    5    6
5    4   1    2    3    4
6    4   1    2    3    4
7    4   8    10   11   12
8    4   7    9    11   12
9    4   8    10   11   12
10   4   7    9    11   12
11   4   7    8    9    10
12   4   7    8    9    10
13   4   14   16   17   18
14   4   13   15   17   18
15   4   14   16   17   18
16   4   13   15   17   18
17   4   13   14   15   16
18   4   13   14   15   16
19   4   20   22   23   24
```

```
20   4   19   21   23   24

21   4   20   22   23   24

22   4   19   21   23   24

23   4   19   20   21   22

24   4   19   20   21   22

25   4   26   28   29   30

26   4   25   27   29   30

27   4   26   28   29   30

28   4   25   27   29   30

29   4   25   26   27   28

30   4   25   26   27   28

adjacent vertices ADJVRT

no. of adjacent vertex pointers

vertex ID, no. of adjacent vertices, adjacent vertex IDs

120

1   3   2   4   5

2   3   1   3   6

3   3   2   4   7

4   3   1   3   8

5   3   1   6   8

6   3   2   5   7

7   3   3   6   8

8   3   4   5   7

9   3   10    12    13

10    3   9    11    14

11    3    10    12    15
```

12	3	9	11	16
13	3	9	14	16
14	3	10	13	15
15	3	11	14	16
16	3	12	13	15
17	3	18	20	21
18	3	17	19	22
19	3	18	20	23
20	3	17	22	24
22	3	18	21	23
23	3	19	22	24
24	3	20	21	23
25	3	26	28	29
26	3	25	27	30
27	3	26	28	31
28	3	25	27	32
29	3	25	30	32
30	3	26	29	31
31	3	27	30	32
32	3	28	29	31
33	3	34	36	37
34	3	33	35	38
35	3	34	36	39
36	3	33	35	40
37	3	33	38	40
38	3	34	37	39

```
39   3   35   38   40

40   3   36   37   39

face normals FNORMS

no. of faces

face ID,unit normal X,Y,Z

30

1    +0.000000E+00   -1.000000      +0.000000E+00

2    +1.000000      +0.000000E+00   +0.000000E+00

3    +0.000000E+00   +1.000000      +0.000000E+00

4    -1.000000      +0.000000E+00   +0.000000E+00

5    +0.000000E+00   +0.000000E+00   -1.000000

6    +0.000000E+00   +0.000000E+00   +1.000000

7    +0.000000E+00   -1.000000      +0.000000E+00

8    +1.000000      +0.000000E+00   +0.000000E+00

9    +0.000000E+00   +1.000000      +0.000000E+00

10    -1.000000      +0.000000E+00   +0.000000E+00

11    +0.000000E+00   +0.000000E+00   -1.000000

12    +0.000000E+00   +0.000000E+00   +1.000000

13    +0.000000E+00   -1.000000      +0.000000E+00

14    +1.000000      +0.000000E+00   +0.000000E+00

15    +0.000000E+00   +1.000000      +0.000000E+00

16    -1.000000      +0.000000E+00   +0.000000E+00

17    +0.000000E+00   +0.000000E+00   -1.000000

18    +0.000000E+00   +0.000000E+00   +1.000000

19    +0.000000E+00   -1.000000      +0.000000E+00

20    +7.071000E-01   +0.000000E+00   +7.071000E-01
```

```
21   +0.000000E+00   +1.000000    +0.000000E+00

22   -7.071000E-01   +0.000000E+00   -7.071000E-01

23   +7.071000E-01   +0.000000E+00   -7.071000E-01

24   -7.071000E-01   +0.000000E+00   +7.071000E-01

25   +0.000000E+00   -1.000000    +0.000000E+00

26   +1.000000    +0.000000E+00   +0.000000E+00

27   +0.000000E+00   +1.000000    +0.000000E+00

28   -1.000000    +0.000000E+00   +0.000000E+00

29   +0.000000E+00   +0.000000E+00   -1.000000

30   +0.000000E+00   +0.000000E+00   +1.000000
```

END OF GEOM.DAT

III.2.3.2 File WKSTAT.DAT

{The robot and positioning table are specified as general kinematic chains. This specification can be extended to three or more kinematic chains for more complex workstations.}

```
*number of kinematic chains
2
*NUMBER OF DEGREES OF FREEDOM, 'NRDOF(I)'
6  3
*ROBOT LINK PARAMETERS

{LINK - a part attached to the link frame.
JTYPE- 0=revolute, 1=prismatic
ALPHA,LENGTH,THETA,DIST- kinematic parameters.
MIN,MAX- joint limits.}

LINK    JTYPE    ALPHA    LENGTH    THETA    DIST    MIN.    MAX.
0       0        -90.     0.        0        0.      -160.   160.
1       0         0.      432       0        149.5   137.    43.
```

```
2       0       90.     -20.5     90       0.      -52.     -128
0       0      -90.     0.        0.       433     -110.    170..
0       0       90.     0.        0        0.      -100.    100.
3       0       0.      0.        0        56.5    -266.    266.
*TABLE LINK PARAMETERS
LINK    JTYPE   ALPHA   LENGTH    THETA    DIST    MIN.     MAX.
0       0      -90.     0.        0        0       -1000.   1000
0       0       90      0         0        0       -1000.   1000.
4       0       0       0         0        0       -1000    1000
*Workpiece cluster id.
{Identity of parts cluster representing the workpiece.}
5
*INITIAL CONFIGURATION ARRAY ICONF

{The initial solution configuration of the robot.}
{ 0 0 0 0 0 0  L A NF }
{ 0 0 0 1 0 0  L A F  }
{ 0 0 1 0 0 0  L B NF }
{ 0 0 1 1 0 0  L B F  }
{ 1 1 1 0 0 0  R A NF }
{ 1 1 1 1 0 0  R A F  }
{ 1 1 0 0 0 0  R B NF }
{ 1 1 0 1 0 0  R B F  }
{where:  R- right handed arm configuration   }
{        L- left handed arm configuration    }

{        A- elbow above configuration        }
{        B- elbow below config.              }

{        NF- wrist not flipped               }
{        F-  wrist flipped                   }

0   0   1   1   0   0
WORLD COORDINATE TO ROBOT COORDINATE TRANSFORMATION MATRIX 'H00'.
1.   0.   0.   50.
0.   1.   0.   50.
0.   0.   1.   0.
0.   0.   0.   1.
*GRIPPER FLANGE TO TOOL COORDINATE TRANSFORMATION MATRIX 'G6G'.
0   1   0   0
-.707   0.   .707   170.7
0.   0.   0.   1.
*TABLE TO WORKPIECE COORDINATE TRANSFORMATION MATRIX 'HTW'
1   0   0   0
0   1   0   0
0   0   1   10
0   0   0   1
*TABLE TRANSFORMATION 'HPRE'
1   0   0   -600
0   1   0   0
0   0   1   200
0   0   0   1
```

```
*TABLE TRANSFORMATION 'HPOST'
1  0  0  -100
0  1  0  -100
0  0  1  0
0  0  0  1
*TOOL ROTATION PARAMETERS:  MAX. NO. OF STEPS,
ANGLE STEP SIZE (DEGREES)
{These parameters specify the rotation of the torch about its axis.}
18   10.

{The first record is for the robot, the second record is for the table.}
*Maximum allowable joint value changes from one location to the next.
(degrees)
15. 15. 15. 15. 15. 15.
5. 5. 10.
*Minimum joint value changes during joint interpolated motion.(degrees)
1. 1. 1. 1. 1. 1.
1. 1. 2.
*Maximum joint velocities.  (degrees/sec)
82. 53. 122. 227. 241. 227.
20. 20. 20.

END OF WKSTAT.DAT
```

III.2.3.3 File SETUP.DAT

```
{The part ids listed in this file must exist, or errors will occur.  Care

must be taken to change this file if a new workpiece or different

workstation is introduced.}


workstation parts to be tested for interference, number, part
15
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
workstation parts to be shown, wnumb,wkpiec
15
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
additional parts to be shown, rnumb,robotg
{This is a list of parts to be shown during replay only}
0

END OF SETUP.DAT
```

III.2.4 Running AUTOWELD. To execute AUTOWELD, type 'autoweld'. After starting AUTOWELD, the following message will appear:

```
< DO YOU WISH TO INITIALIZE FILES? (T/F) >
```

If a partial or complete program has been previously generated, or if some user-defined seams have been created, then the AUTOWELD environment can be restored to its previous state by answering 'f' or 'F'. If you wish to initialize all files and start from scratch, respond 't' or 'T'. Note: to restore the previous environment, the files generated by the previous run of AUTOWELD must exist unaltered, and the workpiece and workstation specifications must not have changed. After responding, AUTOWELD will be loaded and started. This takes about 30 seconds.

An image will appear on the screen. On the right hand side is an information display. Important information is displayed here, as well as a current menu of user commands. An area at the top of the screen will display messages from AUTOWELD to the user. At the center of the screen will be a yellow wire-frame image of the workstation. The CAD seams will be shown in red. The user defined seams are shown in magenta. Welded seams are shown in green. The world coordinate frame is shown for reference.

III.2.4.1 Exiting AUTOWELD. Pressing PF1 at any time will cause AUTOWELD to terminate before the next frame update. The current state will be automatically saved in files so that AUTOWELD can resume from the saved state.

III.2.4.2 Mouse modes. There are two mouse modes, switched by pressing <PF2>. Initially, mouse mode 1 is enabled.

a. Mouse mode 1.

- Changing viewpoint

Moving the mouse cursor horizontally while pressing <left mouse> will change the view azimuth. Moving the cursor vertically will change the view inclination.

- Moving the positioning table

Moving the cursor horizontally while pressing <middle mouse> will cause one joint of the positioning table to change. Joint one is the default. Other joints are selected by pressing the corresponding joint number on the main keyboard.

b. Mouse mode 2

- Zooming and moving window

Moving the cursor to the right while pressing <left mouse> will zoom in on the image. Moving the cursor to the left will zoom out. While <middle mouse> is pressed, the view window will follow the cursor in any direction.

III.2.4.3 Resetting state. All currently defined welds can be removed from the image and from the program by pressing <PF3>. If there are no welds, then all currently defined seams are deleted.

III.2.4.4 Replaying the welding program. The current welding program can be simulated or replayed at any stage by pressing <PF4>. The program is replayed from the start to the end. The replay can be cancelled by pressing <PF4> again. During replay, the user has several options:

a. skip section - skip to the start of the next path. The program will skip to the start of the next weld or inter-weld path following the current path.

b. double step size - double the size of the joint increments used in generating a joint interpolated motion. This is useful for increasing the speed.

c. halve step size - halve the size of the joint increments. This is useful for increasing the resolution for interference detection.

The following information is shown on the screen during replay:

a. welding parameter settings - the welding parameters selected by the expert welder are shown while a weld is being replayed.

b. elapsed time - the following elapsed times are saved and displayed:

   - welding time for current seam
   - total welding time
   - time for current inter-weld motion
   - total inter-weld time
   - total elapsed time from start of program.

III.2.4.5 Global switches. These global switches are enabled in all modes:

a. Solid/wire-frame switch

Initially, AUTOWELD is in the wire-frame graphics mode. Pressing <ENTER><1> switches AUTOWELD to the shaded solid mode.

Pressing <ENTER><1> again restores the wire-frame mode. In the solid mode, objects in the workstation are shown as shaded solids, with hidden surfaces removed. AUTOWELD runs much more slowly in this mode.

b. CAD seam highlighting

Normally, the CAD seam is shown by red dots at each defined point, with green dots at nodes. For increased visibility, the seam can be highlighted by pressing <ENTER><2>. The CAD seam is then shown as a red herringbone with green crosses at the nodes.

c. Exact separating distances

Normally, objects are tested only for interference. If desired, the exact separating distance and connecting vector for all pairs of parts can be calculated and shown on the screen. This mode is activated by pressing <ENTER><3>. The response will slow considerably in this mode.

d. Interference testing

Normally interference is tested every time object in the workstation move. This can be switched off by pressing <ENTER><4>. This is useful for speeding up a motion which is known to be interference free.

e. Interfering paths

When this mode is activated by pressing <ENTER><5>, all interfering paths encountered during a program replay are stored. This is useful for determining which paths interfere without having to test each one individually. This can also be used to verify that the entire program is interference free.

III.2.4.6 Picking. Picking is done identically whenever this guide calls for the user to pick something. First, the mouse is used to move the point of the cursor over a graphical object to be picked. To pick the object, press <right mouse>. In some cases, AUTOWELD requires <right mouse> be pressed once first to activate the picking mode. This insures that the response to the pick will be immediate the second time <right mouse> is pressed.

III.2.4.7 Interactive modes of AUTOWELD

A. Seam definition mode

In this mode, the use defines the seams to be welded by interactively picking points on the CAD reference seam. To enter the seam definition mode, press <1>.

To define a seam, use the cursor to pick points of the CAD reference seam which will define the new seam. The CAD reference seam is shown as a string of red dots, with green dots denoting the nodes. The dots are the defined seam points which are to be picked. To improve the resolution, use zoom. The CAD seam can be highlighted if desired. Pick the desired starting point, the desired end point and any connecting nodes if the seeam contains more than one CAD segment. The points must be picked in the correct sequence. To exit this mode, move the cursor to an empty area and pick twice. The new seam will be displayed on the screen, with an arrow to indicate its orientation.

B. Weld definition mode

In this mode, the user controls the generation of weld trajectories, and makes modifications to the weld and seam data. To enter the weld definition mode, press <2>. The user has a number of options which are listed on the screen:

a. quit - exit this mode

b. remove seam - the next user defined seam which is picked will be removed

c. reverse direction - the direction of the next seam picked will be reversed

d. unweld seam - the next welded seam picked will be unwelded and the weld trajectory will be removed from the program

e. show order - the order of the welded seams will be shown

f. move table - the table will be moved during this weld.

g. initial position - store the current table position as the initial position

h. final position - store the current table position as the final position.

If none of the above modes are active, then AUTOWELD will attempt to generate a feasible trajectory for the next seam picked. If a feasible trajectory is found, it will be entered into the program and the seam will be coloured green on the screen. Otherwise, the user will be advised to reposition the workpiece and try again.

C. Inter-weld path mode

This mode allows the user to simulate, test and modify inter-weld paths. To enter the inter-weld path mode, press <3>.

The user must now specify the inter-weld path to be considered. This is done by picking the weld which follows the desired path. If the desired path is the one following the last weld, then the direction can be reversed by pressing <right arrow> before picking. Now the picked weld preceeds the desired path.

Once an inter-weld path is selected, the robot moves to the first position along the path. The user has a number of options which are listed on the screen:

a. quit - exit this mode

b. increment position - step to the next inter-weld position

c. edit path - enter path modification mode

d. double step size - double the size of the steps during joint-interpolated motion.

e. halve step size - halve the size of the steps

f. single step mode - move a single joint increment

g. next interfering path - jump to the next inter-weld path in the stored list of paths with interference

h. move forward - move forward between positions

i. move backward - move backward between positions

Press <right arrow> to start joint-interpolated motion between the current and next position. Press it again to stop the motion. In the single step mode, the motion stops after one joint increment.

Similarly, press <left arrow> to reverse the direction.

D. Path modification mode

This mode is entered from the inter-weld path mode to modify the inter-weld path. The user has the following options:

a. quit - exit this mode

b. move robot - move a selected robot joint

c. move table - move a selected positioning table joint

d. add position - add a new position after the current position

e. remove position - remove the current position

f. move position - relocate the current position

g. single step - move a single increment at a time

h. increase joint - increase the selected joint value

i. decrease joint - decrease the selected joint value

To move a joint, select the robot or the table, then press the joint number on the keyboard. The joint value will be continuously increased or decreased until the key is pressed again. In single step mode, the joint moves just one increment each time the key is pressed.

III.3 SUBROUTINE DESCRIPTIONS

The following is a list of the subroutines called by AUTOWELD, with a brief description of each and a list of links to other subroutines. The source code for each of these subroutines is located in a file of the same name. For example, subroutine 'name' is in file 'name.f'.

main: This is the main program.

calls: drscrn,displl, drseam,drsequ,drwire,drwkst,dfpath, getnpt,initfm, initgr, init, input, lsort, mksequ, mkpath, proces, rdmodl, smpick, store, times, trandb

addpos: Add a new position to an inter-weld path.

calls: none

called from: editpa

adjcnt: Find the vertices or faces adjacent to a given vertex or face.

calls: ajface, ajvrts

called from: cdist, finsep

arclen: Calculate the arclength of a seam.

calls: none

called from: proces

calcs: Calculate weld parameter correction factors to account for gravity.

calls: none

called from: exprt2

cdist: Find the vertex of a convex polyhedron which is closest to (or farthest from) a given plane.

calls: adjcnt, getcrd

called from: finbes

check: Check if a picked seam has already been welded.
calls: none
called from: smpick

config: Systematically try all robot solution configurations until a feasible solution is found.
calls: worpos
called from: genpat

cvhull: Generate the convex hull of the projection of a convex polyhedron in screen space.
calls: ajvrts, fplist, getvrt, getcrd, screen, vrtlst
called from: disjnt

dfpath: Interactively define an inter-weld path.
calls: editpa, posobs
called from: main

disjnt: Test if the projections of two convex polyhedra in screen space are disjoint.
calls: cvhull
called from: ohtest

displl: Generate an information display on the screen.
calls: setmen
called from: intest, main

drpart: Draw a polyhedral part as a shaded solid.

calls: drpoly, facvrt, fplist, getcrd, getnml, getvrt, vrtlst

called from: drwkst

drpoly: Draw a filled polygon.

calls: getvrt, getcrd

called from: drpart

drscrn: Draw a set of world coordinate axes on the screen

calls: none

called from: intest, main

drseam: Generate graphical objects for the CAD reference seam.

calls: none

called from: main

drsequ: Define graphical objects representing user-defined seams.

calls: none

called from: main, mksequ, revdir

drtext: Write a graphical text message on the screen.

calls: none

called from: editpa, genpat, input, mkseam, smpick drwire: Draw a wire-frame depiction of the scene.

calls: fplist, facvrt, getnml, getvrt, getcrd, vrtlst

called from: drwkst, intest, main

drwkst: Draw a complete depiction of the workstation scene including seams and welds.

calls: drpart, drwire, plist, psort

called from: main

edges: Find the shortest vector connecting an edge of one convex polyhedron to an edge of the closest face of a second polyhedron.

calls: etest

called from: finbes

edgvrt: Find the shortest vector connecting a point to an edge.

calls: dotprd

called from: etest

editpa: Interactively modify an inter-weld path.

calls: addpos, movpos, rempos

called from: dfpath

etest: Find the shortest vector connecting two edges.

calls: dotprd, edgvrt, solve, unit, xprod

called from: edges

evpick: Evaluate the results of a 'pick'.

calls: none

called from: mkseam

exprt1: Generate weld parameter values as a function of fixed, CAD welding specifications.

calls: none

called from: wgeom

exprt2: Generate weld process parameter values which are a function of the seam orientation with respect to the gravity vector.

calls: none

called from: wparam

finbes: Given two convex polyhedra, find a plane tangent to one of them such that the minimum distance from the plane to the other polyhedron is maximized. If the polyhedra are disjoint, this is the separating plane. The distance and separating vector are returned.

calls: ajvrts, cdist, edges, ftest, getvrt, getcrd, getnml, pface, vrtlst

called from: plist

finsep: Find a face separating two convex polyhedra.

calls: adjcnt, facvrt, getnml, getcrd, getbes, putbes

called from: plist

ftest: Test if a point on a face plane lies inside the boundaries of the face.

calls: getvrt, getcrd, vrtlst

called from: finbes

genpat: Generate a robot welding trajectory and calculate welding parameters for a given seam.

calls: config, drtext, wgeom, wparam

called from: plist, trandb

hmult: Perform a matrix multiplication using the IRIS hardware.

calls: none

called from: worpos, init

init: Initialize the workstation description from files.

calls: hmult, matinv

called from: main

initgr: Initialize the graphics environment

calls: none

called from: main

inpath: Generate a joint-interpolated inter-weld path, and test it for interference.

calls: none

called from: main

input: Detect interactive user input and set appropriate flags.

calls: drtext

called from: main

intest: Test a proposed robot and positioner solution for interference.

calls: drwire, displl, plist, trandb

called from: worpos

joints: Find the explicit inverse kinematic solution for a six degree of freedom robot.

calls: none

called from: worpos

limits: Test the joint solutions against their physical limits.

calls: none

called from: worpos

lsort: Update and re-sort the seam and weld definitions.

calls: none

called from: main

mkseam: Compose a user-defined seam interactively by picking points on the reference CAD seam.

calls: drtext, evpick, picksm

called from: mksequ

mksequ: Generate a sequence of user-defined seams.

calls: drwequ, mkseam

called from: main

movpos: Interactively move an inter-weld position.

calls: open, shut

called from: worpos

mtable: Calculate the transformation relating the positioning table position relative to the world frame.

calls: none

called from: worpos

noap5: For a five degree of freedom robot, calculate the torch transformation given the tip location and the orientation of the torch axis.

calls: nxprod

called from: worpos

norm: Calculate the seam normal vector as a function of the orientation of the adjoining parent surfaces.

calls: none

called from: proces

obsort: Sort a set of convex polyhedra by depth, and calculate their bounding boxes.

calls: ajvrts, fplist, getvrt, getcrd, screen, vrtlst

called from: psort

obswap: Replace the polyhedron at the head of the depth priority list with a polyhedron obscured by the polyhedron currently at the head of the list.

calls: none

called from: ohsurf

ohsurf: Perform a hierarchy of hidden-surface tests to order a set of convex polyhedra for hidden surface removal.

calls: initgp, ohtest, osort2, obswap

called from: psort

ohtest: Perform a hierarchy of hidden-surface tests on a candidate polyhedron to see if it obscures any others.

calls: disjnt, splsid

called from: ohsurf

osort2: Find the first polyhedron in the list which is obscured by the current candidate polyhedron
calls: none
called from: ohsurf

pface: Project a vertex point onto a face plane.
calls: facvrt, getnml, getcrd
called from: finbes

picksm: Pick a seam.
calls: none
called from: mkseam, smpick

plist: Generate separating planes for a set of convex polyhedra.
calls: facvrt, finsep, finbes, getsep, getnml
called from: drpath, drwkst, intest

posobs: Read a set of inter-weld positions from a file.
calls: none
called from: dfpath

proces: Process the raw CAD reference seam data, generating seam locations, tangents, and normals at each point.
calls: arclen, norm, nxprod, tangen
called from: main

psort: Sort a set of convex polyhedra for hidden surface removal.

calls: obsort, ohsurf

called from: drwkst

rdmodl: Read the geometry and topology data file and set up the database.

calls: none

called from: main

rempos: Remove an inter-weld position.

calls: none

called from: editpa

revdir: Reverse the direction of a user-defined seam.

calls: delobj, drsequ

called from: smpick

rmseam: Remove a user-defined seam.

calls: none

called from: smpick

screen: Transform world coords into screen coords. Emulates the 'screen' routine of the IRIS library.

calls: none

called from: cvhull, obsort

setmen:	Set up an area on the screen for an information display. calls: none called from: displl
smpick:	Interactive seam picking and automatic weld path generation. calls: drtext, genpat, picksm, revdir, rmseam, unweld called from: main
splsid:	Test the order of two convex polyhedral parts relative to a separating plane between them. calls: none called from: ohtest, osort2
store:	Store and retrieve welds and seam definitions. calls: none called from: main
tangen:	Calculate the tangent vector for the seam at each point. calls: unit called from: proces
times:	Calculate the elapsed time between adjacent inter-weld positions such that the speed is maximized without exceeding any joint velocity limits. calls: none called from: main

tolrot: Rotate the welding torch about its axis to avoid joint limits and interference.

calls: none

called from: worpos

tran: Transform the database geometry by modelling and viewing transformations.

calls: gtvcrd, gtfnor, tracrd, tranor

called from: trandb

trandb: Transform the reference geometry database into world or screen coordinates.

calls: fclist, getpts, tetsep, ptvlst, tran, transp

called from: drpath, intest, main2, mkpath, posobs

transp: Transform the known separating planes.

calls: none

called from: trandb

unweld: Unweld a previously welded seam.

calls: none

called from: smpick

wgeom: Calculate geometric weld parameters based on the CAD welding specifications. This information is required before the weld trajectory can be calculated, so the seam orientation is not an available parameter.

calls: exprtl

called from: genpat

worpos: Generate a feasible robot welding trajectory which satisfies joint limit and avoids interference.

calls: hmult, intest, joints, limits, mtable, noap5, nxprod, tolrot

called from: config

wparam: Calculate welding process parameters after the trajectory is specified. The seam orientation can be used as a factor in these calculations.

calls: exprt2

called from: genpat

III.3.1 Subroutine Source Libraries.

III.3.1.1 Math library. The following subroutines are contained in file 'math.f'.

det2x2, det3x3, dotprd, matdet, matinv, nxprod, sol3x3, solve, unit, vdiff, flen, xprod

det2x2: Find determinant of 2x2 matrix.

det3x3: Find determinant of 3x3 matrix.

dotprd: Find dot product of two vectors.

i2: A function which converts integer to integer*2 values.

matdet: Find the determinant of a 3x3 matrix.

matinv: Invert a 4x4 homogeneous transformation.

nxprod: Normalized cross product.

sol3x3: Solve a 3x3 set of linear equations.

solve: Find the intersection points of the common normal of two lines.

unit: Normalize a vector.

vdiff: Find the difference of two vectors.

vlen: Find the length of a vector.

xprod: Find cross product of two vectors.

III.3.1.2 *Frame coherence library*. The following subroutines are contained in file 'fm2fm.f':

initfm, getbes, putbes

initfm: Initialize the previous frame data at startup.

getbes: Get the previous best face and closest vertex for a given pair of parts.

putbes: Store the new best face and closest vertex for a given pair of parts.

III.3.1.3 *Geometry data base management library*. The sources for the following subroutines are contained in the file 'dbms.f':

ajface, ajvfac, ajvrts, facvrt, fclist, fplist, getcrd, getnml, getnpt, getpts, getvrt, gtfnor, grvcrd, initgp, ptvlst, tracrd, tranor, vrtlst.

ajface: Find the faces adjacent to a face.

ajvfac: Find the faces adjacent to a vertex.

ajvrts: Find the vertices adjacent to a vertex.

facvrt: Return the coordinates of the first vertex of a given face.

fclist: Return the starting and ending face indices for a given cluster.

fplist: Return the starting and ending face indices for a given part.

getcrd: Return the coordinates for a given vertex pointer.

getnml: Return the surface normal for a given face.

getnpt: Return the total number of parts.

getpts: Return the first part and the number of parts for a given cluster.

getvrt: Return the vertex pointer for a given vertex index.

gtfnor: Return the untransformed face normal for a given face.

gtvcrd: Return the untransformed first vertex for a given face.

initgp: Initialize the global parameters.

ptvlst: Return the starting and ending vertex ids for a cluster.

tracrd: Transform a given vertex.

tranor: Transform a given normal.

vrtlst: Return the starting and ending vertex indices for a given face.

III.3.2 Subroutines which call the IRIS graphics library.

The following subroutines call the IRIS graphics library. These subroutines must be modified if a different graphics system is used.

main, displl, drpart, drpoly, drscrn, drseam, drsequ, drtext, drwire, drwkst, editpa, genpat, hmult, initgr, input, intest, mkseam, picksm, plist, screen, setmen, smpick, tran, trandb, transp.

III.4 DATA STRUCTURES

III.4.1 External Data Files. The following are the data files used by AUTOWELD.

Internal name	Actual name
cad	CAD.DAT
cadlst	CADLST.DAT
corect	CORECT.DAT
geom	GEOM.DAT
ilocat	ILOCAT.DAT
itable	ITABLE.DAT
joint	JOINT.DAT
plocat	PLOCAT.DAT
procad	PROCAD.DAT
ptable	PTABLE.DAT
save	SAVE.DAT
setup	SETUP.DAT
weltem	WELTEM.DAT
wkstat	WKSTAT.DAT
wpcdat	WPCDAT.DAT
wpoint	WPOINT.DAT
wptemp	WPTEMP.DAT

wtable	WTABLE.DAT

III.4.1.1 File descriptions

Input files

cad: CAD seam reference data.

cadlst: pointer file for 'cad'

geom: workstation geometry and topology

setup: specification of parts to be tested and shown

wkstat: workstation parameters and kinematic specifications

wpcdat: CAD seam welding parameters

Intermediate files

joint: joint angles for a weld path

procad: processed seam data

weltem: speed, torch orientation along a weld path

wptemp: seam orientation along weld path relative to gravity vector

Output files

corect: vectors lateral and normal to weld for real time correction

ilocat: inter-weld robot and table positions

itable: pointers into 'ilocat'

plocat: robot and table joint angles for a welding path

ptable: pointers into 'plocat'

wpoint: pointers into 'wtable'

wtable: welding process parameters

III.4.1.2 File specifications

cad: unformatted, existing, direct access

records: r(3), n1(3), n2(3)

r(3) - seam locations

n1(3), n2(3) - unit normals of adjacent surfaces

These coordinates are specified relative to the workpiece reference frame.

Opened by: proces

cadlst: unformatted, existing, direct access

record i: nstart, npoint

nstart - seam i starting record in 'cad'

npoint - the number of records for seam i in 'cad'

Opened by: drseam main mksequ proces worpos

corect: unformatted, direct access

record i: enorm(3), elat(3)

enorm(3) - unit vector normal to weld at point i

elat(3) - unit vector lateral to weld at point i

These vectors are specified relative to the world reference frame.

Opened by: main worpos

geom: formatted, existing, sequential

This file is created and modified by an external solid modelling program, and is read by subroutine 'rdmodl'.

Opened by: rdmodl

ilocat: unformatted, direct access

record i: robot joint values, positioning table values, time increment between previous and current position.

Opened by: addpos dfpath inpath main movpos rempos times unweld worpos

itable: unformatted, direct access

record i: robot joint values, positioning table values, time increment between previous and current position.

Opened by: main worpos

ptable: unformatted, direct access

record 1: number of welds

record i+1: first and last position records in 'plocat' for weld i

Opened by: main unweld worpos wparam

save: formatted, sequential

Contains specifications for previously defined seams and welds.

Opened by: store

setup: formatted, sequential

Created with text editor and read by subroutine 'init'

Opened by: rdmodl

weltem: unformatted, sequential

record i: speed, alpha, beta, stkout

speed - welding speed

alpha - longitudinal torch angle

beta - transverse torch angle

stkout - torch gap

Opened by: wgeom worpos

wkstat: formatted, sequential

Created with the editor, and read by subroutine 'init'.

Opened by: init

wpcdat: unformatted, direct access

record i: joint type, cross sectional area, thickness 1, thickness 2, seam type (straight or curved)

Opened by: wgeom worpos wparam

wpoint: unformatted, direct access

record i: pointer into 'wtable' corresponding to position i along weld

Opened by: main wparam

wtemp: unformatted, sequential

record i: t(3), a(3) for point i

t(3) -	seam tangent
a(3) -	seam normal

Opened by: worpos wparam

wtable: unformatted, direct access

record i: voltage, wire feed rate, gas flow rate, current for location i along weld

Opened by: main wparam

III.4.2 Internal Data Structures.

III.4.2.1 Geometry and Topology Data Structure. The geometry and topology of objects in the workstation are defined by a boundary representation of vertices and faces. Redundant information is included to speed processing.

Definitions:

Cluster -	A set of convex polyhedral parts which are fixed relative to some common reference frame. A cluster can be a non-convex object composed of convex parts.
Geometry -	Specification of vertex coordinates and face normals of a set of parts relative to some fixed reference frame.
Part -	A convex polyhedral solid approximating an actual object.
Topology -	A specification of the shapes of objects, independent of actual dimensions. A topological description specifies the number of vertices, edges and faces, and their relationships to each other.

III.4.2.1. Description of Fortran Variables.

AJVERT(*) - linear list of vertices adjacent to vertices

The entries are ordered contiguously, by increasing vertex id. Each record contains a vertex id.

CLUSTR(6,*) - list of clusters

The cluster id corresponds to the array index. Each record contains the following information about the corresponding cluster:

first part id

number of parts

first face index

number of faces

first vertex index

number of vertices

FACES(4,*) - list of faces

The faces are ordered contiguously, by increasing part id (and consequently by increasing cluster id). Each record contains the following information:

first vertex index (pointer into VERTS)

number of vertices

first adjacent face index (pointer into FACES3)

number of adjacent faces

FACES2(*) - linear list of faces adjacent to vertices

The faces are ordered by increasing vertex id. Each record contains a face id.

FACES3(*) - linear list of faces adjacent to faces

The faces are ordered by increasing face id. Each record contains a face id.

FNORMS(3,*) - table of face normals

The indices of this array correspond to FACES. Each record contains the x, y, z components of the face normal.

OVERTS(4,*) - pointers to vertices and faces adjacent to vertices

The array index corresponds to the vertex id. Each record contains the following information:

first adjacent face (pointer into FACES2)
number of adjacent faces
first adjacent vertex (pointer into AVJERT)
number of adjacent vertices

PARTS(2,*) - list of parts

The part id corresponds to the array index. The parts are ordered contiguously, by increasing cluster id. Each record contains the following information about the corresponding part:

first face index
number of faces

TCOORD(3,*) - table of current, transformed vertex coordinates
Each record contains the transformed x, y, z coordinates of the corresponding vertex.

TNORMS(3,*) - table of current, transformed face normals
Each record contains the transformed x, y, z components of the face normals.

VCOORD(3,*) - table of untransformed, reference vertex coordinates
Each record contains the untransformed x, y, z components of the corresponding vertex.

VERTS(*) - list of vertices
The vertex ids are ordered contiguously by increasing face id. Vertex ids are repeated for each new face which contains them, so the size of this list is greater than the total number of unique vertices. Each record contains a vertex id, which corresponds to the index of the vertex coordinates in VCOORD, TCOORD.

The number of entries in each array are stored as a set of constants.

Array	Number of entries
AJVERT	NVERT
CLUSTR	NCLUST
FACES	NFACE
FACES2	NFACE2

FACES3	NFACE3
FNORMS	NFACE
OVERTS	NVERT
PARTS	NPT
TCOORD	NVERT
TNORMS	NFACE
VCOORD	NVERT
VERTS	NVL

III.4.2.2 Seam and Weld Data

The CAD reference seam is defined as a single graphical object. Each seam segment is identified by a tag, and each digitized point by an offset. A weld seam is defined by the user by picking points on the CAD seam.

TAGS(3,*) - generated by a pick.

TAGS(1,i) -	ith tag picked
TAGS(2,i) -	first offset from tag i
TAGS (3,i) -	last offset from tag i

A user-defined seam is constructed by sorting a series of pick results and generating the appropriate series of CAD segments comprising the seam.

FILES(3,*) - pointers into a table of seam segments.

FILES(1,i) -	the ith CAD segment
FILES(2,i) -	the first point within the segment
FILES(3,i) -	the last point within the segment

A sequence of user-defined seams can be built up using the following data structures.

SEQUEN(2,*) - pointers into a table of seam segments.

SEQUEN(1,i) - pointer to first segment of seam i.

SEQUEN(2,i) - the number of segments

SEAMS(3,*) - this is a linear array of 'files'.

SEAMS(1,i) - id of ith CAD segment

SEAMS(2,i) - the first point within the segment

SEAMS(3,i) - the last point within the segment

A graphical object is defined for each seam.

GROBJS(*) - an array of graphical object ids for each seam.

Seams which have been picked and welded are stored in a list.

IPICK(*) - a list of seam ids which have been picked and welded.

Each time a seam is picked, it is added to the list IPICK.

If the seam is subsequently welded, it remains in the list. Otherwise, it is removed after processing.

III.4.2.3 Separating Plane Data Structures. A separating plane is defined to be any plane which separates two convex polyhedra into disjoint half spaces. Invariant separating planes exist for polyhedra which are fixed relative to a common reference frame, or whose relative motion is constrained. These planes can be specified apriori, and need not be calculated.

A separating plane is associated with every pair of parts. Thus, the number of separating planes is the square of the number of parts. This number is halved if the order of two parts is irrelevant. The separating planes are ordered as follows:

2/1 3/1 3/2 4/1 4/2 4/3 5/1 5/2 5/3 5/4 ...

where 2/1 signifies a separating plane attached to part 2 and separating parts 2 and 1 into disjoint half-spaces.

Note: by this scheme, separating planes cannot be attached to the part with the lower numerical id. This is a restriction in some cases where constrained relative motion is allowed between the parts.

The separating plane variables are the following:

CSPLAN(*) -	list of pointers to known separating planes
FNLIST(3,*) -	table of transformed separating plane normals
FVLIST(3,*) -	table of transformed separating plane reference points
NCSEP -	the number of known separating planes
NLIST(3,*) -	table of untransformed separating plane normals
PTABLE(*) -	table of pointers to part boundaries in the separating plane sequence
VLIST(3,*)	table of untransformed separating plane reference points

III.4.2.4 Hidden surface removal. The following variables are used for hidden surface removal:

BBOX(6,*) - table of bounding box limits for each part
The minimum and maximum x, y and z coordinates for each part are calculated in screen coordinates and stored.

OPARTS(*) - the ordered parts list

PINDEX - index of the current head of PTLIST, part P

PTLIST(2,*) - the parts list, sorted by miximum screen depth.
Each record contains: the part id, a flag which is reset when the correspoinding part is eliminated from further tests

QINDEX - the index of the last part which overlaps part P in screen depth

III.4.2.5 Frame coherence. If we assume that a pair of parts have moved only slightly relative to each other, then the best separating face and closest vertex of the parts are probably the same, or close to the previous ones. We can exploit this by saving the best faces and closest vertices of each pair after they are calculated. During the next update, we can use these stored values as starting points. In most cases, the search from these points should be much shorter than if arbitrary starting points were used.

LSTBES(2,*) - array of previous best face and closest vertex ids for every pair

Each pair of parts is assigned a unique index by the following formula:

index = nparts*(part1-1) + part2

The corresponding record contains the following information:

the best separating face of part1

the closest vertex of part2

III.4.3 Common Blocks. The following is a list of the common blocks defined in AUTOWELD, and the subroutines which share them:

1. /boldsm/boldsm

   drwkst input

2. /cad/cad

   main proces

3. /cadlst/cadlst

   drseam main mksequ proces worpos wparam

4. /chmode/chmode

   input main

5. /config/iconf

   config init joints noap5

6. /corect/corect

   main worpos

7. /currnt/currnt

   displl exprt2 main wparam

8. /databs/clustr, parts, faces, verts, vcoord, overts, faces2, faces3, ajvert, fnorms, tcoord, tnorms

   ajface ajvfac ajvrts facvrt fclist fplist getcrd getnml getnpt gatpts getvrt gtfnor gtvcrd inirgp fpvlst tracrd tranor vrtlst

9. /decr/decr

    dfpath inpath input

10. /defwel/defwel

    displl input smpick

11. /direct/forwrd, revers

    dfpath editpa input

12. /edit/edit, add, rem, mov

    dfpath displl editpa input

13. /exprt/volt, wsize, feed, gfr, walpha, beta, stkout

    displl exprt2 main wgeom wparam

14. /flags1/keyon, replay, shdist, pickfl, solid, shrobt

    displl input main smpick

15. /flags2/mvtabl, tinit, tfinal

    displl input main smpick worpos

16. /flags3/remove, direv

    input smpick

17. /ftrans/ftran1, ftran2

    drwkst intest main mtable picksm trandb worpos

18. /geom/geom

    main rdmodl

19. /grobjs/grobjs

    drwkst intest main

20. /ident/ident

    drtext main mtable trandb

21. /idseam/idseam

    dfpath smpick

22. /idweld/idweld

    dfpath main smpick

23. /ilocat/ilocat

    addpos dfpath inpath main movpos rempos times unweld worpos

24. /incr/incr

    dfpath inpath input

25. /incrml/incrml

    path input

26. /incrm2/incrm2

    input main

27. /intest/intest

    config displl

28. /intflg/intflg

    drwkst main

29. /intset/intset

    displl input main

30. /invers/inho0, inh6a

    init joints worpos

31. /ipick/icount, ipick

    drwkst main

32. /itable/itable

    addpos inpath main movpos posobs rempos times unweld worpos

33. /jlimit/rmin, rmax

    editpa init limits worpos

34. /jmove/jmove

    editpa input

35. /joint/joint

    main worpos

36. /last/lstbes, nparts

    getbes initfm putbes

37. /limits/nclust, npt, nface, nvl, nvert, nface2, nface3, nfnorm, fend, vend, vcend

    ajface ajvfac ajvrts facvrt fclist fplist getcrd getnml getnpt getpts getvrt gtfnor gtvcrd initgp ptvlst tracrd tranor vrtlst

38. /makep/makep

    displl input main smpick

39. maxjv/maxjv

    init times

40. /menul/menul

    displl input main

41. /mincha/mincha

    dfpath init inpath main

42. /mindis/mindis

    displl drwkst main

43. /minmov/minmov

    init main worpos

44. /mkpath/mkpath

    displl input main smpick

45. /movtab/movtab, tstep

    displl main worpos

46. /nchain/nchain

    config dfpath editpa init inpath main times trandb

47. /npoint/npoint

genpat worpos

48. /nrdof/nrdof

addpos config dfpath displl editpa init inpath joints limits main movpos mtable rempos times trandb unweld worpos

49. /nsave/nsave

main mkseam smpick

50. /nseams/nseams

drseam intest main proces

51. /nseq/nseq

drwkst intest main

52. /parts/part, number

intest rdmodl

53. /plocat/plocat

main worpos

54. /pquit/pquit

input main

55. /procad/procad

drseam drsequ main proces wgeom worpos wparam

56. /ptable/ptable

main unwled worpos wparam

57. /reclen/recl

addpos dfpath drseam drsequ inpath main mksequ movpos posobs proces rempos times unweld wgeom worpos wparam

58. /reset/reset

input main picksm

59. /rmweld/rmweld

    input smpick

60. /robotg/robotg, rnumb

    main rdmodl

61. /robot/link, jtype, alpha, len, theta, dist

    init joints mtable noap5 trandb

62. /robpos/h0g

    noap5 worpos

63. /screen/diff1, diff2, diff3, sum1, sum3, vcx, vsx, vcy, vcy, vsy,

    vcz, vsz

    main screen

64. /seamob/seamob, boldob, endobj

    drwkst intest main mkseam

65. /seplan/ptabl, csplan, vlist, nlist, fvlist, fnlist, ncsep getsep

    plist rdmodl splsid trandb transp

66. /setup/setup

    main rdmodl

67. /shord/shord

    drwkst input

68. /skip/skip

    input main

69. /start2/start2

    dfpath editpa main smpick

70. /start/start

    genpat input main

71. /step/step

    dfpath displl editpa input

72. /stopw/stopw

    config worpos

73. /tesint/tesint

    displl drwkst input main

74. /time/time1, time2, time3, time4, time5

    displl inpath main

75. /tjnts/tjnt1, tjnt2, tjnt3

    input main

76. /tolrot/imax, delta

    init tolrot worpos

77. /trans/hpre1, phost1, hpre2, hpost2

    init main mtable noap5 trandb

78. /update/update

    genpat main mksequ revdir rmseam smpick unweld

79. /values/values

    config dfpath displl drwkst intest main worpos

80. /viewpo/left, right, bottom, top

    intest main screen worpos

81. /view/vdist, azim, inc, twist

    dfpath drscrn drwkst intest main picksm plist trandb

82. /weltem/weltem

    main wgeom worpos

83. /winsiz/winsiz

    drscrn drseam intest main picksm screen

84. /wkpiec/wkpiec, wnumb

main rdmodl

85. /wkstat/wkstat

init main wparam

86. /workpc/workpc

init trandb

87. /wpcdat/spcdat

main wgeom worpos wparam

88. /wpoint/wpoint

main wparam

89. /wptemp/wptemp

main worpos wparam

90. /wtable/wtable

main wparam

III.4.4 Dimensioning of Variables. The initial dimensioning of variables sets certain limits on AUTOWELD. If these limits are exceeded during execution of AUTOWELD, unpredictable errors will result. These limits can be changed by redimensioning the variables in the source files and recompiling.

The current dimensions are shown for the important variables, as well as the subroutines containing the variable declaration.

bbox(6,n) Array of bounding boxes for polyhedra

n = maximum number of parts

(same as 'wkpiec') n=100

Subroutines: psort

buffer(n)	Buffer used by IRIS pick to store hits n = three times the maximum number of expected hits Subroutines: mkseam, smpick	n=300
colist(n)	List of inter-weld paths which interfere. n = maximum number of inter-weld paths (same as the number of welds) Subroutines: drwkst, intest	n=100
files(3,n)	List of segments making up a user-defined seam n = maximum number of segments Subroutines: main, mkseam, mksequ, revdir, smpick	n=10
flist(n)	List of faces adjacent to a face n = the maximum number of adjacent faces Subroutines: finsep	n=20
fprev(n)	List of previously tested faces n = the maximum number of previously tested faces Subroutines: finsep	n=50
grobjs(n)	List of graphical objects for seams. n = maximum number of seams Subroutines: drwkst, intest, main	n=100
ipick(n)	List of previously picked seams (seams which have been welded) n = maximum number of welds Subroutines: check, drwkst, main	n=100
lastag(3,n)	Previous 'tags' n = maximum expected number of tags returned by a pick (same as 'tags')	n=10

	Subroutines: mkseam	
lstbes(2,n)	List of best separating faces and closest vertices for each pair of parts as found from last frame update. n = maximum number of part pairs (the number of parts squared) Subroutines: getbes, initfm, putbes	n=10000
oparts(n)	Ordered list of parts for hidden surface removal n = maximum number of parts to be shown (same as 'wkpiec') Subroutines: drwkst	n=100
part(n)	List of parts to be tested for interference n = the maximum number of parts Subroutines: intest, rdmodl	n=100
posit(n)	List of inter-weld positions n = maximum number of inter-weld positions Subroutines: dfpath	n=10
ptlist(n)	List of parts ordered by depth n = same as 'wkpiec' Subroutines: psort	n=100
pverts(n)	List of vertex ids which are projected onto the convex hull in screen space n = maximum number of points on the convex hull Subroutines: cvhull	n=100
robotg(n)	List of parts to be highlighted during program replay n = the maximum number of parts	n=100

	Subroutines: main, rdmodl	
seams(3,n)	List of segments composing user defined seams.	
	n = maximum total number of segments	n=200
	Subroutines: main	
sequen(2,n)	Pointers to user defined seams	
	n = maximum number of user defined seams	n=100
	Subroutines: main	
sfiles(3,10)	Same as 'files'	
	n = same as 'files'	n=10
	Subroutines: mkseam	
slistl(2,n)	List of screen coordinates for the convex hull of the projection of a convex polyhedron	
	n = the maximum number of points on the convex hull	
	Subroutines: disjnt	n=100
slist2(2,n)	Same as 'slistl'	
	n = same as 'slistl'	n=100
	Subroutines: disjnt	
slist(2,2,n)	Combination of 'slistl', 'slist2' into a single array	
	Subroutines: disjnt	n=10
tags(3,n)	List of tags returned by pick.	
	n = maximum expected number of tags	n=10
	Subroutines: mkseam	
vlist(n)	List of vertices adjacent to a vertex	
	n = maximum number of adjacent vertices	n=20
	Subroutines: cdist, cvhull, finbes, finsep, obsort	
vprev(n)	List of previously tested vertices	

	n = maximum number of previous vertices	n=50
	Subroutines: cdist, finbes, finsep	
wkpiec(n)	List of parts to be shown graphically	
	n = the maximum number of parts	n=100
	Subroutines: main, rdmodl	

III.4.4.1 Geometry Data Base Variables. To change these, edit the file 'common.com'. This file is included to all the subroutines in the module 'dbms.f'. For the change to be made, delete the object file 'dbms.j' before recompiling AUTOWELD.

ajvert(n)	List of adjacent vertices	
	n = maximum sum of all vertices adjacent to all vertices	n=5000
clustr(6,*)	List of cluster information.	
	n = maximum number of clusters	n=100
faces(4,n)	List of face information	
	n = maximum total number of faces	n=1000
faces2(n)	List of faces adjacent to vertices	
	n = maximum sum of all faces adjacent to all vertices	n=5000
faces3(n)	List of faces adjacent to faces	
	n = maximum sum of all faces adjacent to all faces	n=5000
fnorms(4,n)	Table of untransformed face normals	
	n = maximum total number of faces (same as 'faces')	n=1000

overts(4,n)	Pointers to adjacent vertices in 'ajvert'	
	n = maximum number of unique vertices	n=1000
parts(5,n)	List of parts information	
	n = maximum total number of parts	n=100
tcoord(4,n)	Table of transformed vertex coordinates	
	n = maximum number of unique vertices	n=1000
tnorms(4,n)	Table of transformed face normals	
	n = maximum number of face normals	
	(same as 'faces')	n=1000
vcoord(4,n)	Table of untransformed vertex coordinates	
	n = same as 'tcoord'	n=1000
verts(n)	List of vertex ids ordered by faces	
	n = maximum sum of all vertices on all faces	n=5000

III.4.4.2 Separating Plane Variables. To change these dimensions, edit the file 'seplane.com', which is included to the following subroutines:

getsep plist rdmodl splsid trandb transp

Delete the object files for each of these subroutines before recompiling the program

csplan(n)	Table of pointers into separating plane list	
	n = maximum total number of separating planes	
	(half the number of parts squared)	n=5000
fnlist(n)	Transformed separating plane normals	
	n = same as 'csplan'	n=5000
fvlist(n)	Untransformed separating plane normals	

	n = same as 'csplan'	n=5000
nlist(n)	Transformed separating plane reference points n = same as 'csplan'	n=5000
ptabl(n)	Table of known separating planes n = maximum total number of known separating planes	n=5000
vlist(n)	Untransformed separating plane reference points n = same as 'csplan'	n=5000

III.5 SUGGESTED IMPROVEMENTS

A number of useful features and improvements have been identified but not incorporated into AUTOWELD.

1. Add a file handling module to allow a library of workstations, workpieces and programs to be maintained and selectively loaded.
2. Revise the known separating plane data structure so that a separating plane can be defined relative to either one of a pair of parts.
3. Extend the geometric modelling method to include representations of plates or sheets as planar regions and rods as line segments for faster processing. Modify the interference detection and hidden surface removal algorithms appropriately.
4. Calculate and store the convex hulls of clusters. These can be tested against each other for interference. Individual parts need only be tested if this test fails. The algorithm will be improved dramatically for the case of clusters with many parts if their convex hulls are disjoint.

5. Incorporate consistency checking of the geometric data base.
6. Extend AUTOWELD to the case of redundant degrees of freedom, and incorporate higher level path planning using optimization and AI techniques.

APPENDIX IV - CAD FILE GENERATION MODULE PROGRAM GEM

IV.1 CADFILE GENERATION MODULE PROGRAM GEM - ANALYSIS AND PROGRAM DESIGN

The major requirements of the program are to:

a. create a library of parts that a user can select from in an interactive manner to define a workstation including robot and workpiece;

b. allow a user in an interactive manner to define all the clusters, parts and separating planes in a workstation including the robot and workpiece and then automatically create the datafile in form of GEOM.DAT used by AUTOWELD as the geometric CAD datafile;

c. allow a user to correct or add to a previous workstation description.

The topology of the geometric description of the workstation is described in Section 3.1.1. This topology consists of clusters, parts, faces and vertices. Program GEM must automatically generate the vertex and face vector descriptions from the general cluster and part descriptions entered by the user. These descriptions are to be automatically placed in the output datafile in the form of lists.

We will present the description of GEM in the following order.

a. description of reference frames, vectors, equations and list of analysis program variables;

b. flow chart of program(s) and program and data structures;

c. list of program variables.

IV.1.1 Reference frames, vectors, variables and equations. The reference frames for the world, cluster and parts are shown in Figure IV.1.

The main analysis symbols for the above Figure and subsequent analysis are given below.

F_i = ith cluster reference frame.

F_w = world reference frame

F_{ij} = reference frame of jth part on ith cluster

H_{ij} = homogeneous transformation from F_{ij} to F_i

H_{wi} = homogeneous transformation from F_i to F_w

N_λ = λth face normal

n_{ijm} = unit normal, relative to Fij, of mth face on jth part of ith cluster

$\underline{P}_{ij}$ = position vector origin of F_{ij}, relative to F_i

$\underline{R}_{ij}$ = dimensions of part j in ith cluster

$\underline{V}_q$ = qth vertex position relative to Fw

$\underline{X}_i$, $\underline{Y}_i$, $\underline{Z}_i$ = basis unit vectors for F_i, ith cluster reference frame; $\underline{Z}_i$ along line of axis of θ_{i+1}, $\underline{X}_i$ along line of common normal between $\underline{Z}_i$, $\underline{Z}_{i-1}$.

$\underline{\rho}_{ijk}$ = position vector, relative to F_{ij}, of kth vertex of jth part of ith cluster

$\underline{\mu}_{ln}$ = separating plane unit normal for separating plane between parts l and n ($l<n$)

$\underline{\nu}_{ln}$ = separating plane point for separating plane between parts l and n ($l<n$)

The indices and limits are defined as

$K = 1,2 \ldots K_{ij}$ = kth vertex on jth part on ith cluster
$m = 1,2 \ldots M_{ij}$ = mth face on jth part on ith cluster
$j = 1,2 \ldots J_i$ = jth part on ith cluster
$i = 1,2 \ldots I$ = ith cluster

However, we ultimately need one global numbering series for clusters, one for parts, one for vertices, one for normals and one for separating planes. These global indices are given below.

$i = 1,2 \ldots I$ = cluster index and numbering series

$\eta = 1,2 \ldots H$ = part index and numbering series

$$H = \sum_{i=1}^{I} J_i$$

$q = 1,2 \ldots Q$ = vertex index and numbering series

$$Q = \sum_{i=1}^{I} \sum_{j=1}^{J_i} K_{ij}$$

$\lambda = 1,2 \ldots \Lambda$ = face index and numbering series

$$\Lambda = \sum_{i=1}^{K} \sum_{j=1}^{J_i} M_{ij}$$

$s = 1,2 \ldots S$

Note that I and J_i are set by the user and K_{ij} and M_{ij} are specified automatically in the part library specifications.

IV.1.2 Equations. The equations defining the vertex position and face normal vectors and the above state indices are given below noting that $\underline{\rho}_{ijk}$ and $\underline{\eta}_{ijm}$ are given from the parts library subroutine and $\underline{\xi}_{ij}$, γ_{ij}, $\underline{\zeta}_{ij}$ and $\underline{\rho}_{ij}$ are user specified.

The transforms Hwi and Hij are found from

$$\underline{H}_{wi} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\underline{H}_{ij} = \begin{bmatrix} \underline{\xi}_{ij} & \underline{\gamma}_{ij} & \underline{\zeta}_{ij} & \underline{\rho}_{ij} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We find $\underline{V}_q$ and $\underline{N}_\lambda$ from

$q = 0$

$q = q + 1$

$\underline{V}_q = \underline{H}_{wi}\underline{H}_{ij}\underline{\rho}_{ijk}(\underline{R}_{ij})$

$k = 1,2 \ldots K_{ij}$

$j = 1,2 \ldots J_i$

$i = 1,2 \ldots I$

$\lambda = 0$

$\lambda = \lambda + 1$

$\underline{N}_\lambda = \underline{H}_{wi}\underline{H}_{ij}\underline{\eta}_{ijm}(\underline{R}_{ij})$

$m = 1,2 \ldots M_{ij}$

$j = 1,2 \ldots J_i$

$i = 1,2 \ldots I$

For the separating planes, $\underline{\mu}_{1n}$, $\underline{\nu}_{1n}$ are set by the user for fixed known separating planes. The equations defining the global indices are

For the jth part in the ith cluster

$$\eta = \sum_{ip=1}^{i-1} J_{ip} + j$$

For the kth vertex and mth face on the jth part of the ith cluster

$$q = \sum_{i_p}^{i-1} \sum_{j_p}^{J_{ip}} K_{ip} + \sum_{j_p=1}^{j-1} K_{ijp} + k$$

$$\lambda = \sum_{i_p}^{i-1} \sum_{j}^{J_{ip}} M_{ipj} + \sum_{j_p=1}^{j-1} M_{ip} + m$$

and to find the global separating plane index

$$s = \sum_{n_p}^{n-1} (n\rho - \ell) + \ell$$

IV.1.3 Geometric Definition of Parts. The face and vertex numbering systems for two parts i.e. a cylinder and a box used in subroutine PART are given below.

IV.1.4 Program Design. The modules that make up the package to create the CAD file for program AUTOWELD are the following:

GEM	- main program
PART	- parts library subroutine
ETVAL	- subroutine to find global part number
QVAL	- subroutine to find global vertex number
LAMVAL	- subroutine to find global face number

The flow charts for program GEM and subroutine PART are given in the following figures.

IV.1.5 Symbol Table for Program GEM and Subroutine PART. The symbols used in coding GEM and PART in Fortran 77 are given below;

Programming Symbol	Analysis Symbol/Description
ANS	prompt decision variable in GEM
DATAFILE	auxiliary datafile name in GEM
ET	η, global part number

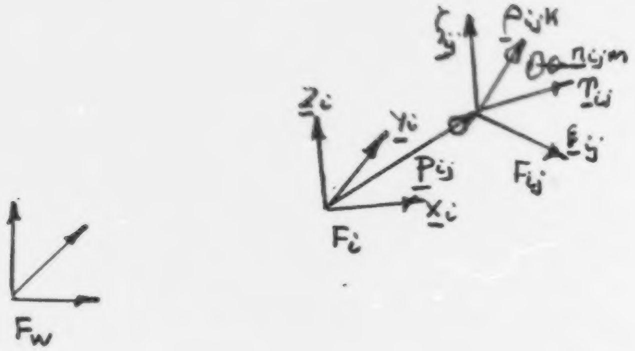

$\underline{\zeta}_{ij}$
$\underline{\rho}_{ijk}$
θ_a
$\underline{\eta}_{ijm}$
$\underline{T}_{ij}$
$\underline{z}_i$
$\underline{y}_i$
$\underline{\xi}_{ij}$
$\underline{P}_{ij}$
F_{ij}
$\underline{x}_i$
F_i
F_w

Figure IV-1. Reference Frames and Vectors

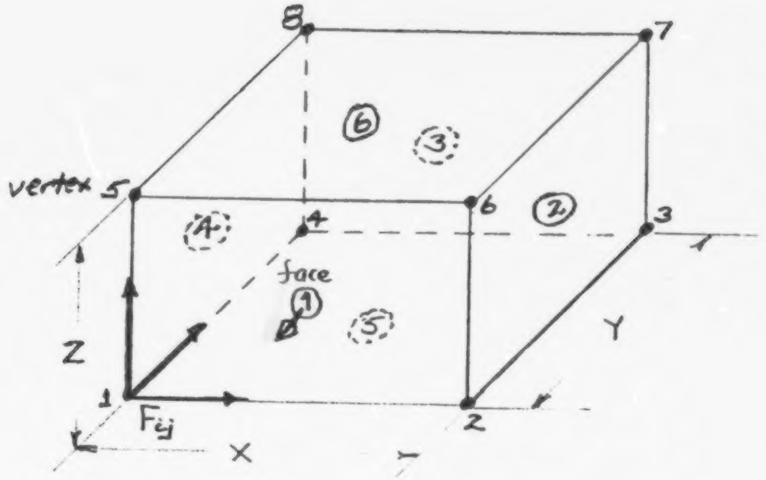

8
7
⑥
3
vertex 5
6
②
4
3
face
①
⑤
Z
Y
1
F_{ij}
2
X
4

Figure IV-2. Box Faces and Normals

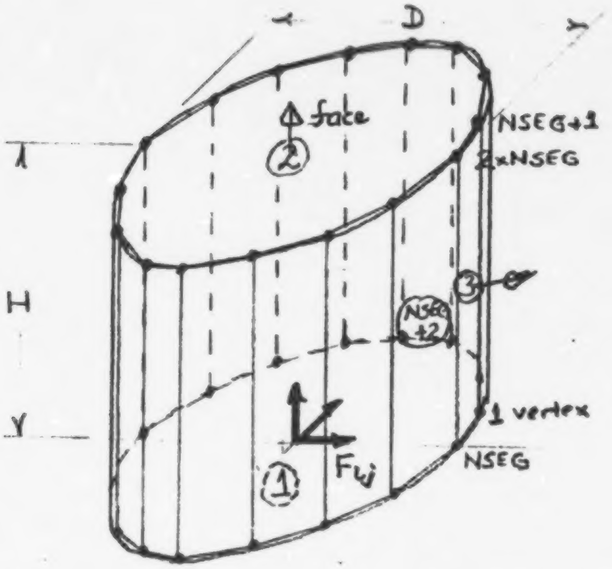

D
face
2
NSEG+1
2×NSEG
λ
H
3
NSEG
+2
γ
1 vertex
NSEG
1
Fij

Figure IV-3. Cylinder Faces and Normals

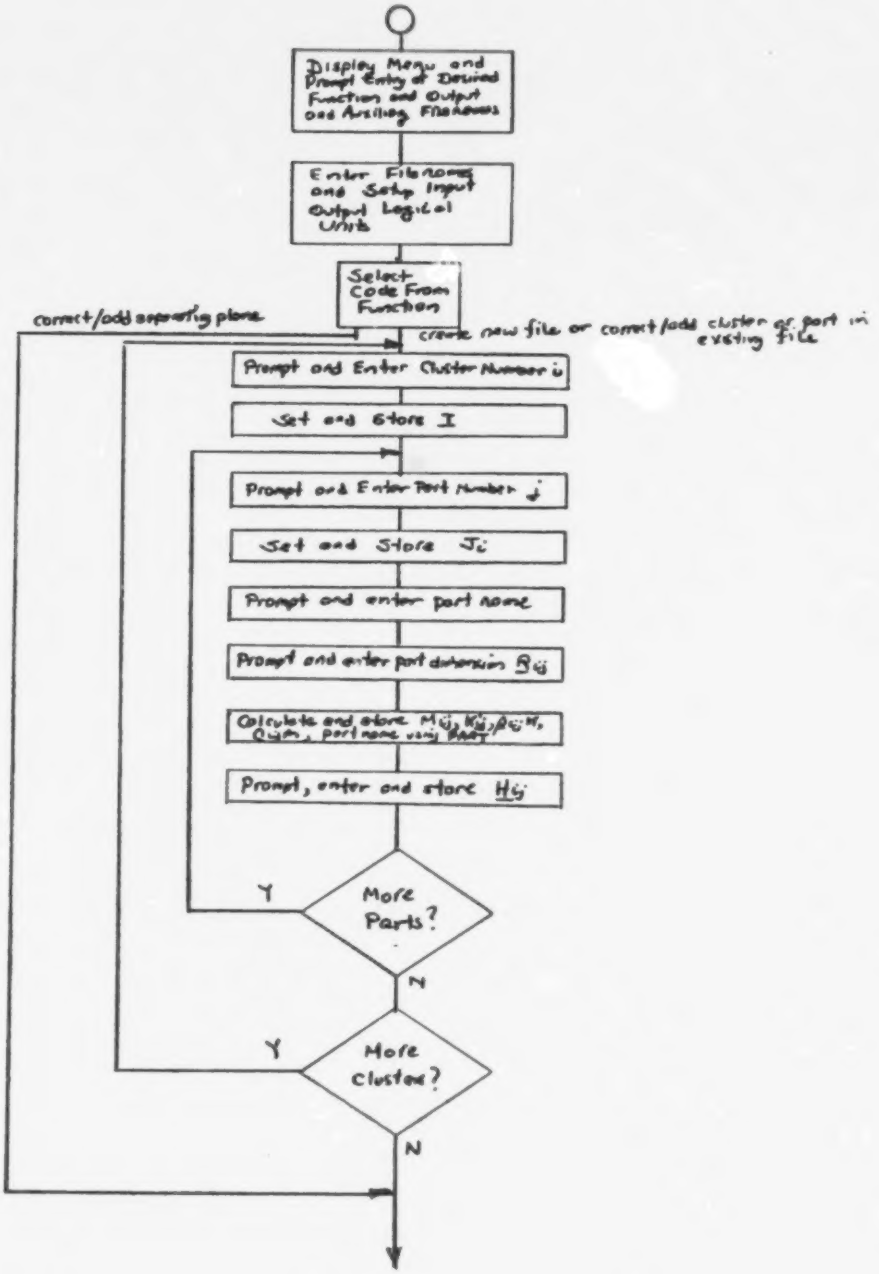

Display Menu and Prompt Entry of Desired Function and Output and Ancillary Filenames
Enter Filenames and Setup Input Output Logical Units
Select Code From Function
correct/add operating plane
create new file or correct/add cluster or part in existing file
Prompt and Enter Cluster Number i
Set and Store I
Prompt and Enter Part number j
Set and Store Ji
Prompt and enter part name
Prompt and enter part dimension Bij
Calculate and store Mij, Kij, βijM, Qijm, part name using PART
Prompt, enter and store Hij
Y
More Parts?
N
Y
More clusters?
N

Figure IV-4. Flowchart for Program GEM

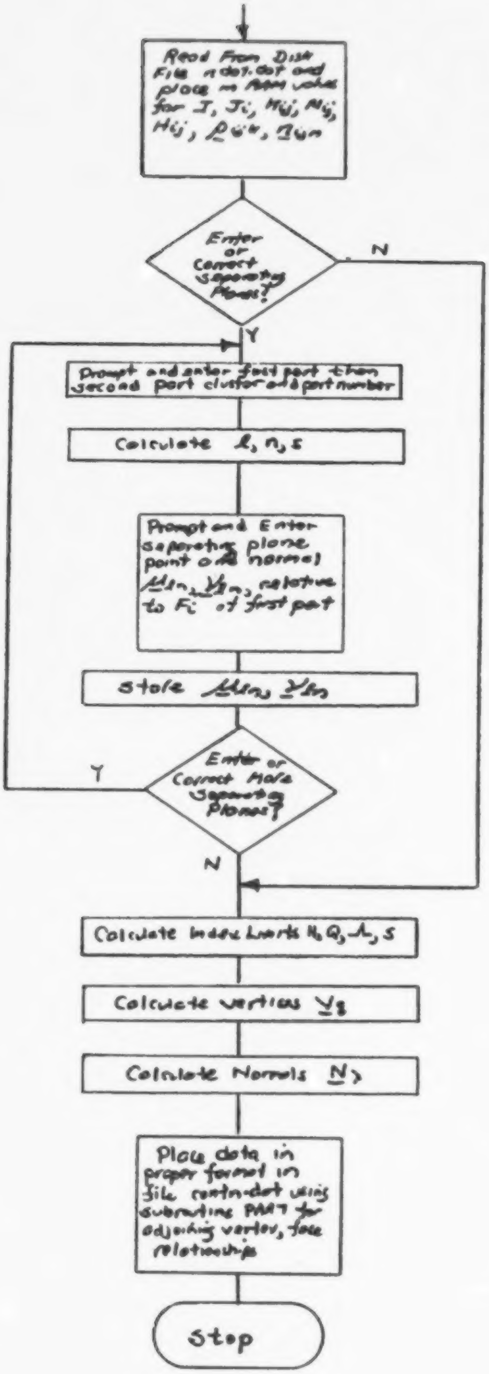

Read From Disk File a dat.dat and place in RAM values for I, Ji, Hij, Nij, Hij, Ri, Iin
Enter or Correct Separating Planes?
N
Y
Prompt and enter first part then second part cluster and part number
Calculate l, n, s
Prompt and Enter separating plane point and normal Uin, Vin, relative to Fi of first part
store Uin, Vin
Enter or Correct More Separating Planes?
Y
N
Calculate Index Limits N, Q, λ, S
Calculate vertices Vs
Calculate Normals Ns
Place data in proper format in file contn.dat using subroutine PART for adjoining vertex, face relationships
Stop

Figure IV-5. Flowchart for Program GEM(continued)

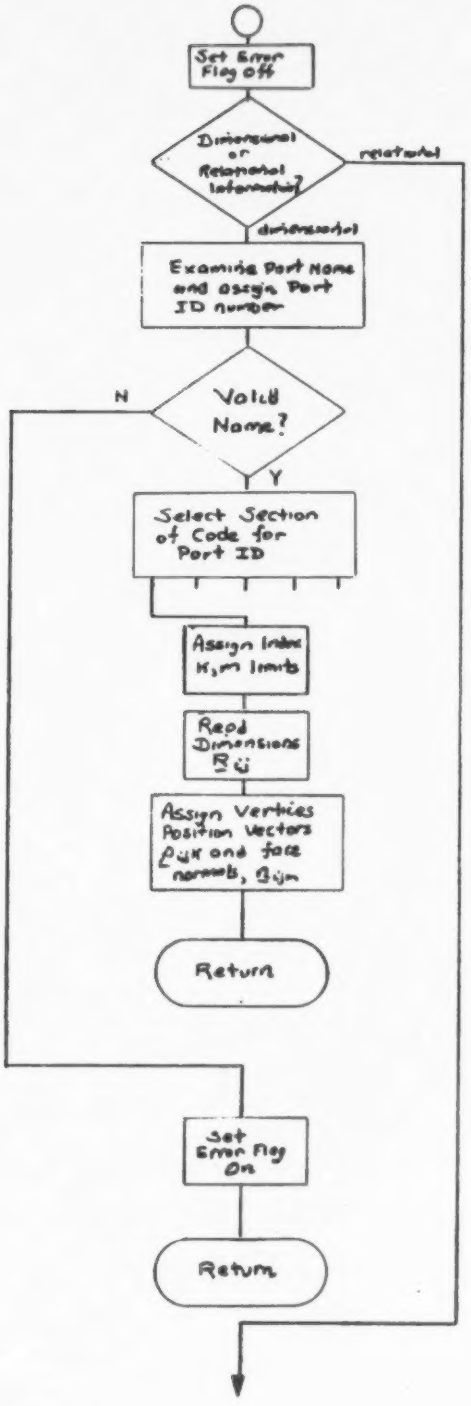

Set Error Flag off
Dimensional or Relational Information?
relational
dimensional
Examine Part Name and assign Part ID number
N
Valid Name?
Y
Select Section of Code for Part ID
Assign Index K,m limits
Read Dimensions B_{ij}
Assign Vertices Position Vectors ρ_{ijk} and face normals, β_{ijm}
Return
Set Error Flag On
Return

Figure IV-6. Flowchart for Subroutine PART

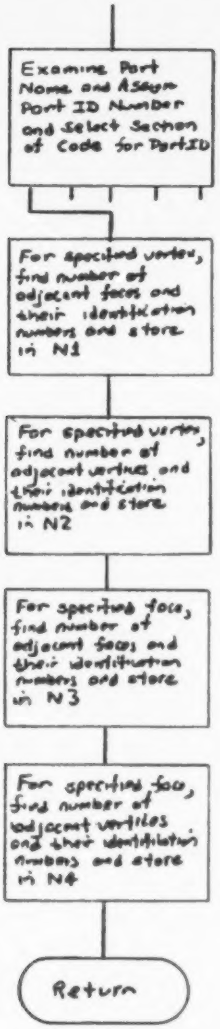

Examine Part Name and Assign Part ID Number and Select Section of Code for PartID
For specified vertex, find number of adjacent faces and their identification numbers and store in N1
For specified vertex, find number of adjacent vertices and their identification numbers and store in N2
For specified face, find number of adjacent faces and their identification numbers and store in N3
For specified face, find number of adjacent vertices and their identification numbers and store in N4
Return

Figure IV-7. Flowchart for Subroutine PART(continued)

FILENAME	GEM output file name
H	H, total number of parts
HT(I,J,II,JJ);II,JJ=1..4	$\underline{H}$ij
II	I, number of clusters
I,IP	i,ip
IEF	error flag in subroutine PART
IDP	idp
IP1,JP1,IP2,JP2	cluster and part numbers for sep planes in GEM
J,JP	j,jp
JJ(I)	Ji
K	k
KA,KK(I,J)	K,Kij
KMAX	maximum value of Kij allowed in GEM
L,LP	ℓ, ℓp
LS	λ
LMC	Λ
M	m
MA,MM(I,J)	M,Mij
MU(L,N,II),II 1,2,3	$\underline{\mu}$ln
MMAX	maximum value of Mij allowed in GEM
N	n
NN(λ,II), II=1,2,3	$\underline{N}$λ
NR(I,J,M,II),II=1,2,3	$\underline{n}$ijm
NRP(M,II),II=1,2,3	$\underline{n}$m
NU(L,N,II),II=1,2,3	$\underline{\nu}$ln

NS	selection number
NCL	maximum number of clusters allowed in GEM
NPAR	maximum number of parts in a cluster in GEM
NV	vertex selection number argument in PART
NF	face selection number argument in PART
N1,N2,N3,N4	output arguments in PART giving face / vertex relational information
NSEG	number of cylinder wall segments in PART
NP	part id number in subroutine PART
NPM	number of dimension parameters in Rij
P(I,J,II),II=1,2,3	$\underline{p}_{ij}$
PNAME	part name in subroutine PART
Q	q, global part number
QQ	Q
R(JJ),JJ=1,2..NPM	$\underline{R}_{in}$ for subroutine PART
RO(I,J,K,II),II=1,2,3	$\underline{\rho}_{ijk}$
ROP(K,II),II=1,2,3	$\underline{\rho}_{k}$
S	s, separating plane number
SS	S, number of separating planes entered by user
UP(I,J,II),II=1,2,3	$\underline{\gamma}_{ij}$
VV(Q,II)II=1,2,3	$\underline{v}_{q}$
XI(I,J,II),II=1,2,3	$\underline{\xi}_{ij}$
X,Y,Z	X,Y,Z

IV.2 CADFILE GENERATION MODULE PROGRAM GEM - USERS GUIDE

The function of program GEM is to create a datafile with a name of the form confn.dat (example names are test1.dat, test2.dat) containing a geometric statement of the workstation information in a format suitable for use by program AUTOWELD in file GEOM.DAT.

Program GEM is an interactive program which prompts the user for input information until sufficient information and decisions have been entered to allow the program to complete its operation and create the output file with name of form confn.dat.

The program input parameters are described below:

a. program option - the user enters 1 to create a new file, 2 to add to or correct an existing file or 3 to add separating plane data.

b. the name of the output file - must be in form confn.dat, example names are test1.dat, test2.dat, the name must be entered in apostrophes i.e. 'test2.dat'.

c. the name of an auxiliary file - name must be in form ndat.dat, example names are aux1.dat, aux2.dat; the name must be entered in apostrophes i.e. 'aux2.dat'.

d. the cluster number and part number of the part to be included in the configuration (see Section 3.1.1 of the main report for a definition of the topology of world, clusters, parts, faces, and vertices) example form - 3 for cluster prompt then 1 for part prompt.

e. the name of the part type to be entered - currently only two names, which must be in apostrophes, can be entered i.e. 'box' for a rectangular box and 'cyl' for a right cylinder.

f. the number of dimensions in the part - 3 for 'box' and 2 for 'cyl'

g. the part dimensions - X,Y,Z for 'box' or diameter then height for 'cyl'; all dimensions in mm.

h. position vector of the part reference frame origin relative to the cluster reference frame in mm. in example form 32.5, -77.,86.

i. part reference frame unit basis vectors relative to the cluster reference; these are the first three columns of Hij described earlier in this Appendix; the vectors are entered in three lines in example form

1.,0.,0.

0.,-1.,0.

j. continued data entry options - answer must be given in form 'yes' or 'no'

k. the cluster number and part number of first part for a separating plane specification in example form 3,2.

l. the cluster number and part number of second part for a separating plane specification in example for 5,3.

NOTE: THE SEPARATING PLANES MUST BE ENTERED IN SEQUENTIAL ORDER FROM LOWEST TO HIGHEST WHERE THE CLUSTER NUMBER THEN PART NUMBER OF THE SECOND PART THEN FIRST PART DETERMINES THE ORDER; AS WELL AT LEAST ONE SEPARATING PLANE MUST BE ENTERED WHEN A NEW FILE IS BEING CREATED I.E. THE PROGRAM OPTION SELECTED WAS 1.

m. a point lying on the separating plane entered in a single line in example form 35.,0.,10.

n. a unit normal to the separating plane entered in single line in example form 1.,0.,0.; the normal points from the second part to the first part and both the point and normal are relative to the cluster frame of the first part.

The output of the program is a file with name of form confn.dat (the actual name was entered by the user) which can be directly copied into file GEOM.DAT for use by program AUTOWELD. Note that the auxiliary datafile format is such that it requires a lot of storage space to allow for subsequent additions to the workstation description. A change in format of this file to reduce its storage requirements would be a useful extension to this work.

APPENDIX V - SOFTWARE SOURCE LISTINGS

V.1 AUTOWELD SOURCE LISTINGS

V.2 GEM SOURCE LISTINGS

V.3 GENPRO SOURCE LISTING

V.4 FACE SPLITTING SOURCE LISTINGS

V.5 DUNCAN METHODS SOURCE LISTINGS

V.6 FORTRAN AND C BENCHMARK PROGRAMS

APPENDIX V- SOFTWARE SOURCE LISTINGS

APPENDIX V.1- AUTOWELD Source Listings

```
      PROGRAM main
C

C     THIS MAIN PROGRAM IS FOR THE AUTOMATIC PROGRAMMING OF A
C     WELDING ROBOT.
C
C     INPUTS: NONE
C
C     OUTPUTS: NONE
C
C
C     SUBROUTINES CALLED:    drscrn,drseam,drsequ,drwkst,drwire,displl,
c                             dfpath,getnpt,
c                             init,initgr,initfm,input,inpath,lsort,mksequ,
c                             mkpath,proces,rdmodl,store,smpick,times,trandb

c     Iris graphics calls: color,clear,frontb,gflush,gexit,pushat,pushma,
c                          pushvi,polarv,popatt,popmat,popvie,swapbu,
c                          setlin,textpo,tpoff,viewpo,window

c     DEFINITION OF VARIABLES:

c     botto-  bottom limit of graphical viewport
c     changm- .true. if a modelling change occurs
c     changr- .true. if the robot is moved
c     changt- .true. if the positioner is moved
c     changv- .true. if a viewing change occurs
c     chmode- .true. to change mouse button functions
c     currnt- welding current
c     diffl,diff2,diff3- parameters for subroutine SCREEN
c     endpat- .true. at the end of a path during replay
c     files-  a reordered array of seam segments (see 'seam')
c     first-  .true. at the start of program replay
c     ftable- .true. to initiate the next inter-weld path during replay
c     ftran2- same as 'how'
c     grobjs- array of graphical object ids
c     how-    homogeneous transformation relating workpiece frame to world
c             frame
c     icount- the number of welded seams
c     ident-  a 4x4 identity matrix
c     ilast-  the id of the last interfering inter-weld path
c     inc-    inclination angle of viewing transformation POLARV
c     incrml- .true. to jump to next inter-weld position
c     intcnt- the number of inter-weld paths which interfered during replay
c     inter-  .true. if an inter-weld path is being replayed
c     intflg- .true. if interference was found along an inter-weld path
c     intlst- an array of interfering paths found during replay
c     intset- .true. to automatically save interfering paths during program
c             replay
c     ioldx,ioldy-  previous cursor position values
c     ipick-  list of welded seams
c     iseam-  the seam id
c     keyon-  .true. for interactive seam definition
c     left-   left limit of graphical viewport
c     makep-  flag used for inter-weld path definition
c     menul-  menu selection flag
```

```
c       mkpath- flag used for inter-weld path definition
c       movtab- array containing initial and final table positions
c       mvtabl- .true. for synchronous positioner motion
c       nchain- the number of kinematic chains
c       nfiles- the number of segments in array 'files'
c       noise-  a noise threshold
c       nrdof-  array containing the number of degrees of freedom of each
c               kinematic chain
c       npt-    the number of parts
c       nsave- the number of graphical objects to be saved during a pick
c       nseams- the number of seams
c       nseq-   the number of seams in a sequence
c       pickfl- .true. when picking mode is active
c       pquit-  .true. to stop replay before it is complete
c       prstop- .true. at the end of program replay
c       recl-    an array containing record lengths of direct access files
c       replay- .true. to replay the robot program
c       reset-  .true. to remove all welds, then to remove all defined seams.
c       right-  right boundary of graphical viewport
c       seam-   array of seam segment ids and pointers into reference
c               CAD seam data
c       sequen- array of pointers to seam segments in 'seam'
c       shdist- .true. to calculate exact separating distances
c       skip-   .true. to skip to the start of the next stage during replay
c       solid-  .true. for shaded, hidden-surface removed graphics
c       start-  .true. if the graphical image is to be reconstructed
c       start2- .true. at start of a joint interpolated move between two
c               inter-weld positions.
c       sum1,sum2,sum3- parameters for subroutine SCREEN
c       tesint- .true. to test for intersection
c       time1-  elapsed time to the current point on the weld from the start
c               of the weld
c       time2-  total elapsed welding time
c       time3-  elapsed time of current inter-weld path
c       time4-  total elapsed inter-weld time
c       time5-  total elapsed program time
c       tinit,tfinal-  .true. to select initial and final table positions
c       tjnt1,tjnt2,tjnt3- flags to enable selective user control of positioner
c                           joints
c       top-    top limit of graphical viewport
c       twist-  twist angle for viewing transformation POLARV
c       update- .true. to update the seam and weld data
c       values- array of robot and positioner joint values
c       vdist-  viewing distance for POLARV
c       vcx,vsx,vcy,vsy,vcz,vsz- parameters for subroutine SCREEN
c       vsave-  temporary storage of 'values'
c       winsiz- array of graphical window limits
c       wsize-  the welding wire diameter
c
c
C*******************************************************************
c
$include /usr/include/fdevice.h
$include /usr/include/fgl.h

c       File names.
```

```
      CHARACTER*20 CADLST,WKSTAT
      CHARACTER*20 WPCDAT,GEOM
      character*20 procad,cad
      character*20 ptable,plocat,wpoint,wtable,corect
      character*20 itable,ilocat
      character*20 weltem
      character*20 setup
      character*20 save
      character*20 joint,wptemp

c     Graphics data.

      real winsiz(6)
      integer*2 tleft,tright,tbotto,ttop
      integer*2 y,z,iangle
      integer*2 left,right,bottom,top
      integer*2 data
      real vdist
      integer*2 azim,inc,twist
      real noise
      integer*2 ioldx,ioldy
C
c     Transformation matrices.

      REAL HOT(4,4),HOW(4,4),HTW(4,4),HPRE(4,4),HPOST(4,4)
      REAL HOO(4,4),H6G(4,4)
      real hprel(4,4),hpostl(4,4),hpre2(4,4),hpost2(4,4)
      real ident(4,4)
      real ftranl(4,4),ftran2(4,4)

c     Workstation geometric data.

      integer robotg(100),wkpiec(100)
      integer wnumb,rnumb
      integer pair(2)



c     Workstation kinematics data.

      integer nrdof(3)
      real values(10,3)
      real vsave(10,3)
      real movtab(10,2),tstep(10)


c     Seam and weld data.

      integer sequen(2,100),seams(3,200)
      integer grobjs(100)
      INTEGER IPICK(100)
      integer nseq
      integer seamob,boldob,endobj
      integer files(3,10)
```

```
c       Interference data.

        real mindis
        integer intlst(50)

c       Misc.

        real temp(10,3)
        integer recl(20)


c       Logical flags.

        logical replay,flagl,shdist,changr,changt,changm,changv
        logical pickfl,tinit,tfinal
        logical solid,shrobt,keyon
        logical start
        logical ftable,prstop,first
        logical mvtabl
        logical ptype
        logical pstat
        logical reset
        logical update
        logical ans
        logical inter,endpat
        logical makep
        logical pquit
        logical chmode
        logical defsms,defwel,menul
        logical tesint
        logical intflg
        logical intset
        logical mkpath
        logical incrm2
        logical start2
        logical skip
        logical tjntl,tjnt2,tjnt3

c
        COMMON /CADLST/CADLST
        COMMON /WKSTAT/WKSTAT
        COMMON /GEOM/GEOM
        common /setup/setup
        common /procad/procad
        common /cad/cad
        common /wpcdat/wpcdat
        common /ptable/ptable
        common /plocat/plocat
        common /itable/itable
        common /ilocat/ilocat
        common /wpoint/wpoint
        common /wtable/wtable
        common /corect/corect
        common /weltem/weltem
        common /wptemp/wptemp
        common /joint/joint
        COMMON /WINSIZ/WINSIZ
        COMMON /VIEW/VDIST,AZIM,INC,TWIST
        COMMON /VIEWPO/LEFT,RIGHT,BOTTOM,TOP
        common /screen/diff1,diff2,diff3,sum1,sum2,sum3,vcx,vsx,vcy,vsy,vcz,vs

        common /trans/hpre1,hpost1,hpre2,hpost2
        common /ftrans/ftran1,ftran2
        common /ident/ident

        common /nrdof/nrdof
        common /nchain/nchain
        COMMON /VALUES/VALUES
        common /movtab/movtab,tstep
        common /tjnts/tjnt1,tjnt2,tjnt3
        common /time/time1,time2,time3,time4,time5

        COMMON /EXPRT/VOLT,WSIZE,FEED,GFR,WALPHA,BETA,STKOUT
        common /currnt/currnt

        COMMON /NSEAMS/NSEAMS
        COMMON /WKPIEC/WKPIEC,WNUMB
        COMMON /ROBOTG/ROBOTG,RNUMB
        common /seamob/seamob,boldob,endobj
        common /grobjs/grobjs
        common /nseq/nseq
        common /ipick/icount,ipick
        common /idweld/idweld

        COMMON /MINDIS/MINDIS
        common /reclen/recl

        common /start/start
        common /flags1/keyon,replay,shdist,pickfl,solid,shrobt
        common /flags2/mvtab1,tinit,tfinal
        common /reset/reset
        common /nsave/nsave
        common /update/update
        common /makep/makep
        common /pquit/pquit
        common /chmode/chmode
        common /menu1/menu1
        common /tesint/tesint
        common /intflg/intflg
        common /intset/intset
        common /mkpath/mkpath
        common /incrm2/incrm2
        common /start2/start2
        common /skip/skip
c
c       Set a welding wire size for demonstration.

        wsize=.035

c       Define sizes of file record lengths.
```

```
c      Files: cad,cadlst,wpcdat,procad,ptable,plocat,itable,ilocat,wpoint,
c             wtable,corect

       recl(1)=36
       recl(2)=8
       recl(3)=20
       recl(4)=52
       recl(5)=8
       recl(7)=8
       recl(9)=4
       recl(10)=16
       recl(11)=24

c      initialize global graphics data
c
       left=0
       right=750
       bottom=0
       top=750
       tleft=0
       tright=1023
       tbotto=0
       ttop=100
       twist=0

       winsiz(1)=-100.
       winsiz(2)=100.
       winsiz(3)=-100.
       winsiz(4)=100.
       winsiz(5)=2000.
       winsiz(6)=50000.
       noise=.01

       inc=900
       vdist=16000.

c      Calculate parameters for subroutine 'screen'.
c      For default setdepth (0,1023).

       diff1=winsiz(2)-winsiz(1)
       sum1=winsiz(2)+winsiz(1)
       diff2=winsiz(4)-winsiz(3)
       sum2=winsiz(4)+winsiz(3)
       diff3=winsiz(6)-winsiz(5)
       sum3=winsiz(6)+winsiz(5)

       vcx=(left+right)/2
       vsx=(right-left)/2
       vcy=(botto+top)/2
       vsy=(top-botto)/2
       vcz=511.5
       vsz=511.5


c      Initialize flags.
```

```
      start=.true.
      ftable=.true.
      prstop=.false.
      pquit=.false.
      first=.true.
      shdist=.false.
      mvtabl=.false.
      pickfl=.false.
      tinit=.false.
      tfinal=.false.
      reset=.false.
      chmode=.false.
      menul=.true.
      tesint=.true.
      intset=.false.
      tjntl=.true.
      tjnt2=.false.
      tjnt3=.false.

c     Initialize other variables.

      nsave=24
      nseq=0
      iseam=0
      intcnt=0
      intlst(1)=0
      ilast=0


c     Define file names.

      cadlst='CADLST.DAT'
      wkstat='WKSTAT.DAT'
      geom='GEOM.DAT'
      setup='SETUP.DAT'
      cad='CAD.DAT'
      procad='PROCAD.DAT'
      wpcdat='WPCDAT.DAT'
      ptable='PTABLE.DAT'
      plocat='PLOCAT.DAT'
      itable='ITABLE.DAT'
      ilocat='ILOCAT.DAT'
      wpoint='WPOINT.DAT'
      wtable='WTABLE.DAT'
      corect='CORECT.DAT'
      weltem='WELTEM.DAT'
      joint='JOINT.DAT'
      save='SAVE.DAT'
      wptemp='WPTEMP.DAT'


      write(*,*) 'DO YOU WISH TO INITIALIZE FILES? (T/F)'
      read(*,*) ans

C
C
```

```
C*****  OPEN CAD FILE LIST, 'CADLST'.
C

        open(unit=1,file=cadlst,form='unformatted',access='direct',
     *       status='old',recl=recl(2))
c       Calculate the number of seam segments in the CAD reference seam.

        nseams=0
        irec=0
5       continue
        irec=irec+1
        read(1,rec=irec) nstart,npoint
        if (nstart.eq.0) goto 6
        nseams=nseams+1
        goto 5
6       continue
        close(1)


C*****  GENERATE THE processed SEAM FILES FROM THE SEAM CAD FILES
C
        call proces

C
c       read the data base from the geometry file.

        CALL RDMODL


c       Get the total number of parts and initialize the frame to frame
c       coherence data structure.

        call getnpt(npt)
        call initfm(npt)


c       initialize graphics
c
        call initgr
        call init

c       Define the variable record lengths.

        recl(6)=4
        do 210 i=1,nchain
           recl(6)=recl(6)+4*nrdof(i)
210     continue
        recl(8)=recl(6)


c       Initialize files.

        if (ans) then
```

```
c          Create new, empty files.

           open(unit=1,file=ptable,form='unformatted',access='direct',
     *          status='new',recl=recl(5))
           write(1,rec=1) 0
           write(1,rec=2) 0,0
           close(1)
           open(unit=1,file=itable,form='unformatted',access='direct',
     *          status='new',recl=recl(7))
           write(1,rec=1) 1,2
           write(1,rec=2) 0,0
           close(1)
           open(unit=1,file=plocat,form='unformatted',access='direct',
     *          status='new',recl=recl(6))
           close(1)
           open(unit=1,file=ilocat,form='unformatted',access='direct',
     *          status='new',recl=recl(8))
           write(1,rec=1) (temp(i,1),i=1,nrdof(1)),(temp(i,2),i=1,nrdof(2)),0
           write(1,rec=2) (temp(i,1),i=1,nrdof(1)),(temp(i,2),i=1,nrdof(2)),0
           close(1)
           open(unit=1,file=wpoint,form='unformatted',access='direct',
     *          status='new',recl=recl(9))
           close(1)
           open(unit=1,file=wtable,form='unformatted',access='direct',
     *          status='new',recl=recl(10))
           close(1)
           open(unit=1,file=corect,form='unformatted',access='direct',
     *          status='new',recl=recl(11))
           close(1)

        else

c          Read the previous set of seam and weld definitions from a file.

           call store(1,save,.false.,ipick,icount,seams,sequen,nseq)

        endif



c       Erase the screen.

        call color(i2(0))
        call frontb(.true.)
        call clear
        call frontb(.false.)

        do 1 i=1,4
             ident(i,i)=1.
1       continue

c       Define the graphical objects for the reference seams.

        seamob=1
        endobj=2
        boldob=3
        call drseam(seamob,boldob,endobj)

c       Recreate the graphical objects for previously defined seams and welds.

        do 18 ind=1,nseq
           istart=sequen(1,ind)
           iend=istart+sequen(2,ind)-1
           nfiles=0
           do 19 ind2=istart,iend
              nfiles=nfiles+1
              do 21 i=1,3
                 files(i,nfiles)=seams(i,ind2)
21            continue
19         continue
           call drsequ(files,nfiles,iobj)
           grobjs(ind)=iobj
18      continue

C
c*****  GRAPHICS LOOP

c       UNTIL (qtest().eq.padpfl)
c
10      CONTINUE

        call viewpo(left,right,bottom,top)

c       Get valuator values and assign to variables.

        if (chmode) then

c          Move window.

           if (getbut(mouse2)) then
              winsiz(1)=winsiz(1)-(getval(mousex)-ioldx)/5.
              winsiz(2)=winsiz(2)-(getval(mousex)-ioldx)/5.
              winsiz(3)=winsiz(3)-(getval(mousey)-ioldy)/5.
              winsiz(4)=winsiz(4)-(getval(mousey)-ioldy)/5.

c          Zoom in or out.

           elseif (getbut(mouse3)) then
              vdist=vdist+10.*(getval(mousex)-ioldx)
           endif
        else

c          Change selected positioner joint value.

           if (getbut(mouse2)) then
              if (tjnt1) then
                 val1=val1+getval(mousex)-ioldx
              elseif (tjnt2) then
                 val2=val2+getval(mousex)-ioldx
              elseif (tjnt3) then
```

```
                 val3=val3+getval(mousex)-ioldx
              endif

c          Change viewing direction.

           elseif (getbut(mouse3)) then
              azim=azim+getval(mousex)-ioldx
              inc=inc-getval(mousey)+ioldy
           endif
        endif

           ioldx=getval(mousex)
           ioldy=getval(mousey)


c       Update positioner joint values.

        if (.not.replay.and..not.makep) then

           if (tjnt1) then
               values(1,2)=val1/10.
           elseif (tjnt2) then
               values(2,2)=val2/10.
           elseif (tjnt3) then
               values(3,2)=val3/10.
           endif
        endif


c       Test for changes in the values of the valuators

c               Changes in object positions

        changm=.false.
        changr=.false.
        changt=.false.
        do 101 i=1,10
           do 102 j=1,3
              if (abs(values(i,j)-vsave(i,j)).gt.noise) then
                 changm=.true.
                 if (j.eq.1) then
                    vsave(i,j)=values(i,j)
                    changr=.true.
                 elseif (j.eq.2) then
                    if (.not.getbut(mouse2)) then
                       vsave(i,j)=values(i,j)
                       changt=.true.
                    endif
                 endif
              endif
102        continue
101     continue

c               Changes in viewing parameters

        changv=.false.
```

```
         if (abs(vdist-vdists).gt.noise) then
            vdists=vdist
            changv=.true.
         elseif (abs(azim-azims).gt.noise) then
            azims=azim
            changv=.true.
         elseif (abs(inc-incs).gt.noise) then
            incs=inc
            changv=.true.
         endif


c        Clear screen and prepare for drawing.

         call tpoff
         call viewpo(left,right,bottom,top)
         call color(i2(0))
         call clear
         call color(i2(11))
         call viewpo(left,right,bottom,top)


c        Draw world axes.

         call drscrn


c        Menu display and control section

c        Test the event queue for user input and set flags.

         call input(icount)


c        Save initial and final positions of movable table.

         if (tinit) then
            tinit=.false.
            do 15 i=1,nrdof(2)
               movtab(i,1)=values(i,2)
 15         continue
         elseif (tfinal) then
            tfinal=.false.
            do 16 i=1,nrdof(2)
               movtab(i,2)=values(i,2)
 16         continue
         endif


         if (changm.or.changv.or.start) then

c        Perform viewing and modelling transformations on data base.

            if (solid) then
               call trandb(1,values)
            elseif(changm.or.start) then
```

```
              call trandb(0,values)
           endif

           do 333 i=1,4
              do 334 j=1,4
                 how(i,j)=ftran2(i,j)
334           continue
333        continue

        endif


c*****  WORKSTATION GRAPHICAL DEPICTION



        call window(winsiz(1),winsiz(2),winsiz(3),winsiz(4),winsiz(5),winsiz(6))

        call pushat
        call pushma
        call pushvi

c       Find the next interfering inter-weld path.

        if (mkpath.and.intset) then
           if (incrm2) then
              incrm2=.false.
              intcnt=intcnt+1
              idweld=intlst(intcnt)
              start2=.true.
              if (idweld.eq.0) then
                 intcnt=0
                 mkpath=.false.
                 makep=.false.
                 menu1=.true.
                 start2=.false.
              else
                 makep=.true.
              endif
           endif
        endif

c       Define and edit the inter-weld path.

        if (makep) then
           call dfpath(1,solid,changm)
           if (changm) then
              if (solid) then
                 call trandb(1,values)
              else
                 call trandb(0,values)
              endif
           endif
        endif

        if (getbut(mouse2)) then
```

```
         changm=.false.
      elseif (start) then
         start=.false.
         changm=.true.
      endif

c     Draw the robot, workpiece and table.

      call drwkst(wkpiec,wnumb,shdist,changv,changm,solid)


c     Program replay section.

      if (replay) then

c        Test each inter-weld path and save those which interfere.

         if (intflg) then
            intflg=.false.
            if (iseam.ne.ilast) then
               ilast=iseam
               intcnt=intcnt+1
               intlst(intcnt)=iseam
               intlst(intcnt+1)=0
            endif
         endif

         call pushat
         call pushma
         call color(i2(15))
         call setlin(i2(1))
         if (.not.solid) call polarv(vdist,azim,inc,twist)
         call drwire(0,robotg,rnumb)
         call gflush
         call popatt
         call popmat
      endif

c     For speed, inhibit display update while viewing transformations are
c     being made.

      if (.not.getbut(mouse3)) call displl
      call popvie
      call popmat
      call popatt


      if (keyon) then

c     Interactively define the seams to be welded.

         call mksequ(keyon,nseq,sequen,seams,grobjs)

      else

c        Reset all welds, then all defined seams.
```

```
            if (reset) then
               if (icount.gt.0) then
                  icount=0
               else
                  nseq=0
               endif
                  reset=.false.
                  open(unit=1,file=ptable,form='unformatted',access='direct',
     *                  status='new',recl=recl(5))
                  write(1,rec=1) 0
                  write(1,rec=2) 0,0
                  close(1)
                  open(unit=1,file=itable,form='unformatted',access='direct',
     *                  status='new',recl=recl(7))
                  write(1,rec=1) 1,2
                  write(1,rec=2) 0,0
                  close(1)
                  open(unit=1,file=ilocat,form='unformatted',access='direct',
     *                  status='new',recl=recl(8))
                  write(1,rec=1) (temp(i,1),i=1,nrdof(1)),
     *               (temp(i,2),i=1,nrdof(2)),0
                  write(1,rec=2) (temp(i,1),i=1,nrdof(1)),
     *               (temp(i,2),i=1,nrdof(2)),0
                  close(1)
            endif

c           Enter seam picking mode.
c           The resulting operation depends on the global flags.

            if (.not.mkpath.or..not.intset) then
               call smpick(ipick,icount,nseq,sequen,seams,grobjs)
            endif
         endif


c        Update the seam definition file.

         if (update) then
            update=.false.
            call lsort(ipick,icount,grobjs,sequen,nseq,seams)
         endif



c        Replay the previously generated trajectories by reading the joint
c        angles stored in files.

         if (replay) then

c           Skip to the next weld or next inter-weld path.

            if (skip) then
               skip=.false.
               if (inter) then
                  endpat=.true.
            elseif (.not.prstop) then
               ftable=.true.
            endif
         endif
c
c        Start the program.
c
         if(first) then
            open(unit=11,file=ptable,form='unformatted',access='direct',
     *           status='old',recl=recl(5))
            open(unit=12,file=plocat,form='unformatted',access='direct',
     *           status='old',recl=recl(6))
            open(unit=13,file=wpoint,form='unformatted',access='direct',
     *           status='old',recl=recl(9))
            open(unit=14,file=wtable,form='unformatted',access='direct',
     *           status='old',recl=recl(10))
            read(11,rec=1) nwelds
            if (nwelds.eq.0) pquit=.true.
            first=.false.
            time1=0.
            time2=0.
            time3=0.
            time4=0.
            time5=0.
c
c           Calculate time intervals for the inter-weld paths.
c
            call times(icount)
c
         endif
c
c        Initialize the next weld sequence.
c
         if (.not.pquit) then
         if (ftable) then
            iseam=iseam+1
            read(11,rec=iseam+1) nstart,nend
            if (nstart.eq.0) prstop=.true.
            ftable=.false.
            inter=.true.
            call inpath(1,.true.,iseam,values,endpat)
            volt=0.
            feed=0.
            currnt=0.
            gfr=0.
            time1=0.
         endif
c
c        Replay the inter-weld path. Reset and end the replay if there are
c        no more welds.
c
         if (inter) then
            call inpath(1,.false.,iseam,values,endpat)
            if (endpat) then
               inter=.false.
               if (prstop) then
```

```
                    iseam=0
                    ilast=0
                    intcnt=0
                    replay=.false.
                    first=.true.
                    ftable=.true.
                    prstop=.false.
                    pquit=.false.
                    menul=.true.
                    close(11)
                    close(12)
                    do 110 i=1,10
                       values(i,1)=0.
110                 continue
                 endif
              endif

c          Replay the weld.

           elseif (.not.ftable.and..not.prstop) then
              read(12,rec=nstart) (values(i,1),i=1,nrdof(1)),
     *        (values(j,2),j=1,nrdof(2)),timint
              time1=time1+timint
              time2=time2+timint
              time5=time5+timint
              read(13,rec=nstart) ipoint
              read(14,rec=ipoint) volt,feed,gfr,currnt
              nstart=nstart+1
              if (nstart.gt.nend) then
                 ftable=.true.
              endif
           endif

c       Reset and end replay.

        else

           iseam=0
           ilast=0
           intcnt=0
           replay=.false.
           first=.true.
           ftable=.true.
           inter=.false.
           prstop=.false.
           pquit=.false.
           menul=.true.
           close(11)
           close(12)
           do 120 i=1,10
              values(i,1)=0.
120        continue
        endif
        endif

        call viewpo(left,right,bottom,top)
        call gflush
        CALL SWAPBU
C
        IF(QTEST().ne.padpfl) then
           goto 10
        endif
c       END UNTIL
c*****  END OF GRAPHICS LOOP

c       Store the welding program in a file.

        call store(1,save,.true.,ipick,icount,seams,sequen,nseq)

c       Calculate and store the time intervals between inter-weld positions
c       based on the maximum allowable joint velocities.

        call times(icount)

        CALL GEXIT

        call tpon

        stop
        end
```

```
      subroutine addpos(lunit,idweld,index,values)

c     Add a new position to an inter-weld path.

c     Parameters:     lunit-  logical unit number
c                     idweld- the id of the weld associated with the path
c                     index-  the location of the first position in the
c                             positon table
c                     values- robot and table joint values for a position

c     Subroutines called: none

c     Called from: editpa

c     Variables:

c               nend-   a pointer to the last position record
c               nrdof-  an array containing the degrees of freedom of each
c                       kinematic chain
c               savend- saves the old value of 'nend'
c               savel,save2- temporary storage for 'values'
c               values- the robot and positioner joint values
c

c     Files 'itable', 'ilocat' are read and modified.


      character*20 itable,ilocat
      integer nrdof(3),savend
      real savel(10,3),save2(10,3)
      real values(10,3)
      integer recl(20)


      common /itable/itable
      common /ilocat/ilocat
      common /nrdof/nrdof
      common /reclen/recl


c     Update the position pointers table.

      open(unit=lunit,file=itable,form='unformatted',access='direct',
     *         status='old',recl=recl(7))

      read(lunit,rec=idweld) nstart,nend
      savend=nend
      nend=nend+1
      write(lunit,rec=idweld) nstart,nend

      icount=idweld

10    continue
      icount=icount+1
      read(lunit,rec=icount) nstart,nend
```

```
        if (nstart.ne.0) then
           savend=nend
           nstart=nstart+1
           nend=nend+1
           write(lunit,rec=icount) nstart,nend
           goto 10
        endif

        write(lunit,rec=icount) 0,0

        close(lunit)

c       Add the new position to the positions table.

        open(unit=lunit,file=ilocat,form='unformatted',access='direct',
     *          status='old',recl=recl(8))

        do 20 ind=index+1,savend

           read(lunit,rec=ind) (save1(i,1),i=1,nrdof(1)),
     *     (save1(i,2),i=1,nrdof(2)),time

           if (ind.eq.index+1) then
              write(lunit,rec=ind) (values(i,1),i=1,nrdof(1)),
     *           (values(i,2),i=1,nrdof(2)),time
           else
              write(lunit,rec=ind) (save2(i,1),i=1,nrdof(1)),
     *           (save2(i,2),i=1,nrdof(2)),time
           endif

           do 30 i=1,2
              do 35 j=1,nrdof(i)
                 save2(j,i)=save1(j,i)
35            continue
30         continue

20      continue


           write(lunit,rec=savend+1) (save2(i,1),i=1,nrdof(1)),
     *        (save2(i,2),i=1,nrdof(2)),time

        close(lunit)
        return
        end
        subroutine adjcnt(type,id,list,number,prev,nprev)

c       For a given face (or vertex), return a list of topologically
c       adjacent faces (or vertices) which have not previously appeared.

c       SUBROUTINES CALLED:
c               ajface
c               ajvrts
c
c       Called from:    cdist,finsep
c
c       PARAMETERS:
c               type-   1=face
c                       2=vertex
c               id-     the face or vertex ID
c               list-   a list of adjacent faces (or vertices)
c               number- the number of elements in 'list'
c               prev-   a list of previously encountered faces (or vertices)
c               nprev-  the number of elements in 'prev'
c               ndim1-  the dimension of 'list','temp'
c               ndim2-  the dimension of 'prev'


c********************************************************************************


        integer type,id,number,nprev
        integer list(*), prev(*)
        integer temp(50)
        logical flag


c       add the current adjacent elements to the 'previous' list

        do 10 ind1=1,number
           nprev=nprev+1
              prev(nprev)=list(ind1)
10      continue


c       add the current face or vertex to the head of the list

        number=1
        list(number)=id


c       find the adjacent elements of the current face or vertex

        if (type.eq.1) then
           call ajface(id,temp,numb)
        else
           call ajvrts(id,temp,numb)
        endif
```

```
c       remove those elements that are in the 'previous' list

        do 30 ind1=1,numb
           flag=.true.
           do 40 ind2=1,nprev
              if (flag) then
                 if (temp(ind1).eq.prev(ind2)) flag=.false.
              endif
40         continue
           if (flag) then
              number=number+1
              list(number)=temp(ind1)
           endif
30      continue

        return
        end
```

```
      SUBROUTINE ARCLEN(R,DELS)
C
C     THIS SUBROUTINE CALCULATES ARC LENGTH S.
C
C     INPUTS:
C             R(3,3)=POSITION VECTORS AT POINTS I-1, I, I+1.
C     OUTPUT:
C             DELS=ARC LENGTH FROM POINT I TO POINT I+1.
C
c     Called from: proces

c     Subroutines called: none

C**********************************************************************
C
C
      REAL R(3,3),X,Y,Z,LASTX,LASTY,LASTZ,DELS
      DOUBLE PRECISION DELX, DELY, DELZ
C
      X=R(3,1)
      Y=R(3,2)
      Z=R(3,3)
      LASTX=R(2,1)
      LASTY=R(2,2)
      LASTZ=R(2,3)
      DELX=X-LASTX
      DELY=Y-LASTY
      DELZ=Z-LASTZ
      DELS=DSQRT(DELX**2+DELY**2+DELZ**2)
      RETURN
      END
```

```
        SUBROUTINE CALCS(T,A,CPHI,CTHE)
C
C       THIS SUBROUTINE CALCULATES THE ORIENTATION FACTORS 'CPHI',
C       'CTHE' AS FUNCTIONS OF THE TORCH ORIENTATION RELATIVE
C       TO THE GRAVITY VECTOR.
c       Parameters:
c               T-      seam tangent in world coordinates
c               A-      seam normal in world coordinates
c               cphi,cthe- orientation correction factors
c       Called from: exprt2
c       Subroutines called: none
C
c*******************************************************************************
C
        real T(3),A(3)
        CONV=180./3.14159
C
C*****  'CONV' CONVERTS ANGLES IN RADIAN TO DEGREES
C
C*****  'PHI' IS THE ANGLE BETWEEN +Z (UPWARD) AXIS AND THE TANGENT
C*****  TO THE WELD SEAM
C
        PHI=ACOS(T(3)/SQRT(T(1)**2+T(2)**2+T(3)**2))
C
C*****  'THETA' IS THE ANGLE BETWEEN +Z (UPWARD) AXIS AND THE NORMAL
C*****  TO THE WELD SEAM
C
        THETA=ACOS(A(3)/SQRT(A(1)**2+A(2)**2+A(3)**2))
        PHI=PHI*CONV
C
C*****  DETERMINE ORIENTATION FACTORS 'CPHI' AND 'CTHE' USING -
C*****  THE ORIENTATION ANGLES
C
        THETA=THETA*CONV
        CPHI=5.889*(10.0**(-4))*PHI+0.95
        IF(PHI.GT.90.0) CPHI=-5.89*(10.0**(-4))*PHI+1.05
        CTHE=-5.556*(10.0**(-4))*THETA+1.
        IF(THETA.GT.90.0) CTHE=-1.111*(10.0**(-3))*THETA+1
C
        RETURN
        END
        subroutine cdist(type,normal,fdist,vert,dtest)

c       Find the vertex of a convex polyhedron with the minumum (maximum)
c       distance to a given plane.

c       Called from: finbes

c       SUBROUTINES CALLED:
c               adjcnt
c               getcrd

c       PARAMETERS:
c               type-   1-find the minimum distance
c                       2-find the maximum distance
c               normal- the plane normal
c               fdist-  the plane distance from the origin
c               vert-   receives starting vertex, returns closest vertex
c               dtest-  the minimum (maximum) distance

c       Variables:
c               itest-  .true. if an adjacent vertex is closer (farther) than
c                       the current vertex
c               ntclos- .true. if the current vertex is not the closest
c                       (farthest)
c               nvprev- the number of previously tested vertices
c               startv- .true. when the current vertex is the starting vertex
c               vprev-  a list of previously tested vertices
c               vnumb-  the number of adjacent vertices
c               vlist-  a list of adjacent vertices
c

c*************************************************************************

        real normal(3),fdist,dtest
        real coord(3)
        integer vert,type
        integer vlist(20),vprev(50)
        integer vnumb,nvprev
        logical ntclos,startv
        logical itest


                nvprev=0
                vnumb=0
                startv=.true.
                ntclos=.true.

                if (type.eq.1) then
                   dtest=100000.
                else
                   dtest=-100000.
                endif


c               WHILE (ntclos)
```

```
101          if (.not.ntclos) goto 201

c     find the vertices adjacent to the current vertex
c     and test which one is closest to (farthest from) the face plane.
c     That vertex
c     becomes the new current vertex.

c              Find the adjacent vertices.

               call adjcnt(2,vert,vlist,vnumb,vprev,nvprev)

c              If all the adjacent vertices have already been tested,
c              then the current vertex is the closest (farthest).

               if (vnumb.eq.1) ntclos=.false.

               if (ntclos) then

                 itest=.false.

c                The current vertex is tested only at the start.

                 if (startv) then
                   init=1
                 else
                   init=2
                 endif

c                Test the distances of all the adjacent vertices from the
c                test face.  The one with the minimum (maximum) distance
c                becomes the new current vertex. If none of the adjacent
c                vertices is closer (farther) than the current vertex,
c                then the current vertex is the global minimum (maximum)
c                due to convexity.

                 do 20 ind3=init,vnumb
                   call getcrd(vlist(ind3),coord)
                   vdist=0.
                   do 21 i=1,3
                     vdist=vdist+coord(i)*normal(i)
21                 continue
                   vdist=vdist-fdist

                   if (type.eq.1.and.vdist.lt.dtest.or.
     *                type.eq.2.and.vdist.gt.dtest) then
                     dtest=vdist
                     vert=vlist(ind3)
                     ntclos=.true.
                     itest=.true.
                   endif

20               continue
                 if (.not.itest) ntclos=.false.

               endif
```

```
                          startv=.false.
c                      END WHILE
                       go to 101
201                    continue

        return
        end
```

```
      SUBROUTINE CHECK(IPICK,ICOUNT,IDSEAM,IPREV)
C
C     THIS SUBROUTINE CHECKS IF A SEGMENT HAS ALREADY BEEN
C     PICKED.
C
C     INPUTS: IPICK(100)- ARRAY OF PICKED SEGMENT ID INTEGERS
C
C             ICOUNT- TOTAL NUMBER OF SEGMENTS
C
C             IDSEAM- ID OF SEGMENT TO BE CHECKED
C
C     OUTPUTS: IPREV- 0 IF SEAM NOT PREVIOUSLY PICKED
C                     1 OTHERWIZE
c
c     Called from: smpick
c
c     Subroutines called: none
C
C****************************************************************************************
C
      INTEGER IPICK(100)
C
      IPREV=1
C
      DO 100 I=1,ICOUNT
         IF(IPICK(I).EQ.IDSEAM) GO TO 200
100   CONTINUE
C
      IPREV=0
C
200   CONTINUE
C
      RETURN
      END
```

```
        SUBROUTINE CONFIG(IDSEAM,files,nfiles,ISTAT)
C
C       THIS SUBROUTINE TESTS ALL CONFIGURATIONS OF
C       THE ROBOT UNTIL A FEASIBLE SOLUTION IS FOUND.
C
c       Parameters: idseam- id of current seam to be processed.
c                   files- reference seam ids, and starting and ending points.
c                   nfiles- number of entries in files.
C                   ISTAT- FEASIBILITY STATUS
C                       0=FEASIBLE TRAJECTORY FOUND
C                       1=ROBOT ENVELOPE EXCEEDED
C                       2=ROBOT JOINT LIMIT EXCEEDED
C
C       COMMON BLOCKS CHANGED:  /CONFIG/
C
C       VALUES OF ICONF(6):    (FOR PUMA 560 TYPE ROBOT)
C
C       0 0 0 0 0 0     L A NF          MODIFY CONFIGURATION IN THIS
C       0 0 0 1 0 0     L A F           DIRECTION ONLY
C       0 0 1 0 0 0     L B NF          :
C       0 0 1 1 0 0     L B F           :
C       1 1 1 0 0 0     R A NF          :
C       1 1 1 1 0 0     R A F           :
C       1 1 0 0 0 0     R B NF          V
C       1 1 0 1 0 0     R B F
C
C       WHERE:  R- RIGHT HANDED ARM CONFIGURATION
C               L- LEFT HANDED ARM CONFIG.
C
C               A- ELBOW ABOVE CONFIGURATION
C               B- ELBOW BELOW CONFIG.
C
C               NF- WRIST NOT FLIPPED
C               F- WRIST FLIPPED
C
c       Called from: genpat

C       SUBROUTINES CALLED:    WORPOS

c       Variables:
c               intest- .true. during weld path generation
c               isave-  temporary record of initial configuration
c               nchain- number of kinematic chains
c               nrdof-  number of degrees of freedom
c               save-   temp. record of initial joint values
c               stopw-  .true. if weld generation is aborted
c               values- joint values
c
c
C
C
C*****************************************************************************
C
        INTEGER ICONF(6)
        integer isave(6)
        integer files(3,*),nfiles
        real values(10,3)
        real save(10,3)
        logical stopw
        logical intest
        integer nchain,nrdof(3)

        COMMON /CONFIG/ICONF
        common /stopw/stopw
        common /values/values
        common /nchain/nchain
        common /nrdof/nrdof
        common /intest/intest
C
c       Save the robot and positioner joint variables.

        do 30 i=1,nchain
           do 40 j=1,nrdof(i)
              save(j,i)=values(j,i)
40         continue
30      continue

        intest=.true.

        stopw=.false.

        I=2
        J=2
        K=2

        do 50 ind=1,6
           isave(ind)=iconf(ind)
50      continue
C
100     CONTINUE
C
C*****  TEST CURRENT CONFIGURATION FOR FEASIBILITY
C
           CALL WORPOS(IDSEAM,files,nfiles,ISTAT)

           if (stopw) then
              istat=4
           else
C
C*****  IF A FEASIBLE SOLUTION WAS FOUND, GO TO END
C
           IF(ISTAT.EQ.0) GO TO 200
C
C*****  ELSE TRY A DIFFERENT CONFIGURATION UNTIL ALL POSSIBLE
C*****  CONFIGURATIONS HAVE BEEN TRIED
C
C*****  REVERSE WRIST FLIP CONFIGURATION
C
```

```
            IF(ICONF(4).EQ.0) GO TO 10
               ICONF(4)=0
               GO TO 20
 10         CONTINUE
               ICONF(4)=1
 20         CONTINUE
 C
            I=I-1
            IF(I.GT.0) GO TO 100
 C
            I=2
 C
 C*****  REVERSE ELBOW ABOVE/BELOW CONFIGURATION
 C
            IF(ICONF(3).EQ.0) GO TO 11
               ICONF(3)=0
               GO TO 21
 11         CONTINUE
               ICONF(3)=1
 21         CONTINUE
 C
            J=J-1
            IF(J.GT.0) GO TO 100
 C
            J=2
 C
 C*****  REVERSE LEFT/RIGHT ARM CONFIGURATION
 C
 300        CONTINUE
            IF(ICONF(1).EQ.0) GO TO 12
               ICONF(1)=0
               ICONF(2)=0
               GO TO 22
 12         CONTINUE
               ICONF(1)=1
               ICONF(2)=1
 22         CONTINUE
 C
               K=K-1
            IF(K.GT.0) GO TO 100
            IF(K.EQ.0) GO TO 300

            endif
 C
 200        CONTINUE

 c          If no feasible path was found, restore the previous configuration.

            if (istat.ne.0) then
               do 60 ind=1,6
                  iconf(ind)=isave(ind)
 60            continue
            endif

 c          Restore the robot and positioner joint variables.
```

```
        do 31 i=1,nchain
           do 41 j=1,nrdof(i)
              values(j,i)=save(j,i)
41         continue
31      continue

        intest=.false.
C
        RETURN
        END
```

```
      subroutine cvhull(part,sclist,nverts)

c     Find the convex bounding polygon of a polyhedron projected onto
c     the screen, in screen coordinates.

c     SUBROUTINES CALLED:
c             ajvrts
c             fplist
c             getvrt
c             getcrd
c             screen
c             vrtlst

c     Called from: disjnt

c     PARAMETERS:
c             part-   the polyhedral part to be projected
c             sclist- the convex hull of the projection, in screen coordinates
c             nverts- the number of vertices of the convex hull

c     Variables:

c             coord,coord2-  vertex coordinates
c             lvect-  the current reference vector, tangent to the convex hull
c             pverts- list of previously encountered vertices
c             scoord,scord2-  vertex screen coordinates
c             scsave- the screen coordinates of the current best choice
c                     for the next vertex on the hull
c             vect-   the vector currently being tested against 'lvect'
c             vlist-  vertices adjacent to vertex 'ivert'
c             vsave-  the vector closest to 'lvect' so far
c             xmin-   the minimum x screen coordinate
c

c*****************************************************************************

      integer part,nverts
      integer sclist(2,*)
      integer vlist(20),pverts(100)
      real coord(3),coord2(3)
      integer xmin,first
      integer*2 scoord(3),scord2(3),scsave(3)
      real length,vect(3),lvect(3),vsave(3)
      logical stop,istat


      toler=.00001


c     Get the first vertex of 'part'

      call fplist(part,iface,iend)
      call vrtlst(iface,istart,iend)
      call getvrt(istart,ivert)
```

```
c       Find the vertex with the minimum x screen coordinate.
c       This is the starting vertex.
c       Search recursively through the adjacent vertices of 'ivert' until
c       a vertex is found which has a smaller x screen coordinate than
c       any vertices adjacent to it. This is known to be the global minimum
c       for a convex polyhedron.

        stop=.false.
c       WHILE (.not.stop)
100     if (stop) goto 200
           call getcrd(ivert,coord)
           call screen(1,coord,scoord,istat)
           xmin=scoord(1)
           call ajvrts(ivert,vlist,nvert)
           stop=.true.
           do 10 indl=1,nvert
              call getcrd(vlist(indl),coord2)
              call screen(1,coord2,scord2,istat)
              if (scord2(1).lt.xmin) then
                 xmin=scord2(1)
                 istrtx=scord2(1)
                 istrty=scord2(2)
                 ivert=vlist(indl)
                 stop=.false.
              endif
10         continue
           goto 100
200     continue
c       ENDWHILE

c       Choose the y-axis as the starting reference vector.

        lvect(1)=0.
        lvect(2)=1.

c       Find the second vertex which defines an edge closest to the reference
c       vector. This edge belongs to the convex hull. Make this edge the new
c       reference vector, and its endpoint the new starting vertex.
c       Repeat until the entire convex hull is generated.
c       Store the screen coordinates of the vertices on the convex hull in an
c       ordered list.

        first=ivert
        nverts=0
        stop=.false.
c       WHILE (.not.stop)
101     if (stop) goto 201
           nverts=nverts+1
           pverts(nverts)=ivert
           call getcrd(ivert,coord)
           call screen(1,coord,scoord,istat)
           if (nverts.eq.1) then
              do 250 i=1,2
                 sclist(i,nverts)=scoord(i)
250           continue
           else
```

```
            do 251 i=1,2
               sclist(i,nverts)=scsave(i)
251         continue
         endif
         call ajvrts(ivert,vlist,nvert)
         dotmax=-1.
         do 15 ind2=1,nvert

c           Process only new vertices.

            if (vlist(ind2).ne.pverts(nverts)) then

               call getcrd(vlist(ind2),coord2)
               call screen(1,coord2,scord2,istat)
               length=0.
               do 20 ind3=1,2
                  vect(ind3)=float(scord2(ind3)-scoord(ind3))
                  length=length+vect(ind3)**2
20             continue
               length=sqrt(length)

c              Do not consider the degenerate case when two adjacent
c              vertices are superimposed
c              in screen space.

               if (length.gt.toler) then
                  do 25 i=1,2
                     vect(i)=vect(i)/length
25                continue
                  dotprd=0.
                  do 30 i=1,2
                     dotprd=dotprd+lvect(i)*vect(i)
30                continue
                  if (dotprd.gt.dotmax) then
                     dotmax=dotprd
                     ivert=vlist(ind2)
                     do 35 i=1,3
                        scsave(i)=scord2(i)
35                   continue
                     do 36 i=1,2
                        vsave(i)=vect(i)
36                   continue
                  endif
               endif
            endif
15       continue
         do 40 i=1,2
            lvect(i)=vsave(i)
40       continue

c     Check if the current vertex is already in the vertex list.
c     If so, we have completely specified the convex hull.

         do 300 index=1,nverts-1
            if (.not.stop) then
               if (ivert.eq.pverts(index)) stop=.true.
```

```
                endif
300          continue

          goto 101
201       continue
c         ENDWHILE

          return
          end
```

```
c       This module contains subroutines for interfacing with the geometry
c       data base.

c*****************************************************************************

        SUBROUTINE TRANOR(idim,index,normal)

c       Calculates the transformed surface normals

c       PARAMETERS:
C               index-  the normal index
c               normal- the transformed FNORM vector

c*****************************************************************************

$include common.com

        integer idim,index
        real normal(4,idim)

        do 20 j=1,idim
        ind=index+j-1
        do 10 i=1,3
           tnorms(i,ind)=normal(i,j)
10      continue
20      continue

        return
        end



c*****************************************************************************

        SUBROUTINE TRACRD(idim,index,coord)

c       Insert the transformed homogeneous vertex coordinates into
c       the transformed vertex list.

c       PARAMETERS:
c               index-  the vertex index
c               coord-  the homogeneous vertex coordinates

c*****************************************************************************

$include common.com

        integer idim,index
        real coord(4,idim)

        do 20 j=1,idim
        ind=index+j-1
        do 10 i=1,3
           tcoord(i,ind)=coord(i,j)
10      continue
```

```
20      continue

        return
        end



c*******************************************************************************

        SUBROUTINE FCLIST(iclust,istart,iend)

c       Get the starting and ending face indices for a given cluster of parts.

c       PARAMETERS:
c               iclust- the cluster index
c               istart- the starting face index
c               iend-   the ending face index

c*******************************************************************************

$include common.com

        integer iclust,istart,iend

        istart=clustr(3,iclust)
        iend=istart+clustr(4,iclust)-1

        return
        end



c*******************************************************************************
        SUBROUTINE FPLIST(ipart,istart,iend)

c       Get the starting and ending face indices for a given part.

c       PARAMETERS:
c               ipart-  the part index
c               istart- the starting face index
c               iend-   the ending face index
c*******************************************************************************

$include common.com

        integer ipart,istart,iend

        istart=parts(1,ipart)
        iend=istart+parts(2,ipart)-1
        return
        end



c*******************************************************************************

        SUBROUTINE FACVRT(iface,coords)

c       Return the coordinates of the first vertex of the given face.

c       PARAMETERS:
c               iface-  the face index
c               coords- the coordinates of the first vertex of the face
c*******************************************************************************

$include common.com

        real coords(3)
        integer iface

        do 10 i=1,3
           coords(i)=tcoord(i,verts(faces(1,iface)))
10      continue

        return
        end



c*******************************************************************************

        SUBROUTINE GETCRD(ivert,coords)

c       Get the vertex coordinates for a given vertex.

c       PARAMETERS:
c               ivert-  the index of the vertex
c               coords- the vertex coordinates
c*******************************************************************************

$include common.com

        integer ivert
        real coords(3)

        do 10 i=1,3
           coords(i)=tcoord(i,ivert)
10      continue

        return
        end

c*******************************************************************************
        SUBROUTINE GETNML(iface,normal)

c       Get the surface normal for a given face.

c       PARAMETERS:
C               iface-  the face index
c               normal- the surfac normal

c*******************************************************************************
$include common.com

        integer iface
        real normal(3)

        do 10 i=1,3
           normal(i)=tnorms(i,iface)
10      continue

        return
        end


c*******************************************************************************
        subroutine getnpt(nparts)

c       Returns the total number of parts.

$include common.com

        nparts=npt

        return
        end


c*******************************************************************************
        SUBROUTINE GETPTS(clust,fpart,nparts)

c       Returns the first part and number of parts of a given cluster.

$include common.com
```

```
        integer clust,fpart,nparts

        fpart=clustr(1,clust)
        nparts=clustr(2,clust)

        return
        end


c*******************************************************************************

        SUBROUTINE GTVCRD(idim,index,coord)

c       Return untransformed vertex coordinates.

c       PARAMETERS:
c               index-  the vertex index
c               coord-  the untransformed coordinates

c*******************************************************************************

$include common.com

        integer idim,index
        real coord(4,idim)

        do 20 j=1,idim
        ind=index+j-1
        do 10 i=1,3
           coord(i,j)=vcoord(i,ind)
10      continue
        coord(4,j)=1.
20      continue

        return
        end


c*******************************************************************************

        SUBROUTINE GTFNOR(idim,index,normal)

c       Return untransformed face normals.  The face normals are stored
c       as the vector sum of the first face vertex and the true normalized
c       face normal.

c       PARAMETERS:
c               index-  the normal index
c               normal- the untransformed face normal

c*******************************************************************************
```

```
$include common.com

        integer idim,index
        real normal(4,idim)

        do 20 j=1,idim
        ind=index+j-1
        do 10 i=1,3
           normal(i,j)=fnorms(i,ind)
10      continue
        normal(4,j)=1.
20      continue

        return
        end




c************************************************************************************
        SUBROUTINE GETVRT(ivert,ipoint)

c       Return a vertex pointer.

c       PARAMETERS:
c               ivert-  index into the vertex pointer list.
c               ipoint- the vertex pointer.

c************************************************************************************
$include common.com

        ipoint=verts(ivert)

        return
        end



c************************************************************************************

        SUBROUTINE INITGP

c       Initialize the global parameters.

c************************************************************************************

$include common.com

        fend=nface
        vend=nvl
        vcend=nvert
        return
        end


c*******************************************************************************

        SUBROUTINE PTVLST(iclust,istart,iend)

c       Get the starting and ending vertices for a cluster of parts.

c       PARAMETERS:
c               iclust- the cluster index
c               istart- the starting vertex index
c               iend-   the ending index

c*******************************************************************************

$include common.com

        integer iclust,istart,iend

        istart=clustr(5,iclust)
        iend=istart+clustr(6,iclust)-1

        return
        end


c*******************************************************************************

        SUBROUTINE VRTLST(iface,istart,iend)

c       Get the starting and ending vertices for a given face.

c       PARAMETERS:
c               iface-  the face index
c               istart- the starting vertex index
c               iend-   the ending vertex index.

c*******************************************************************************

$include common.com

        istart=faces(1,iface)
        iend=istart+faces(2,iface)-1

        return
        end


C*******************************************************************************
c*******************************************************************************
```

```
c       THIS SECTION CONTAINS ROUTINES FOR FINDING ADJACENCY.
c*******************************************************************************
c*******************************************************************************
        subroutine ajface(id,list,number)

c       For a given face ID, return a list of adjacent faces.

c       PARAMETERS:
c               id-     the face ID
c               list-   a list of adjacent faces
c               number- the number of elements in the list

c*******************************************************************************
$include common.com


        integer id,list(*),number

        istart=faces(3,id)-1
        number=faces(4,id)

        do 10 i=1,number
           list(i)=faces3(istart+i)
10      continue

        return
        end




c*******************************************************************************
        subroutine ajvrts(id,list,number)

c       For a given vertex ID, return a list of adjacent vertices.

c       PARAMETERS:
c               ID-     the vertex ID
c               list-   a list of adjacent vertices
c               number- the number of elements in the list

c*******************************************************************************
$include common.com


        integer id,list(*),number

        istart=overts(3,id)-1
        number=overts(4,id)
        do 10 i=1,number
           list(i)=ajvert(istart+i)
10      continue

        return
        end




c*******************************************************************************

        subroutine ajvfac(id,list,number)

c       For a given vertex ID, return a list of adjacent faces.

c       PARAMETERS:
c               id-     the vertex ID
c               list-   a list of adjacent faces
c               number- the number of elements in the list
c*******************************************************************************

$include common.com


        integer id,list(*),number

        istart=overts(1,id)-1
        number=overts(2,id)

        do 10 i=1,number
           list(i)=faces2(istart+i)
10      continue

        return
        end
        subroutine dfpath(lunit,changm)

c       Interactively define an inter-weld path. The user can step through the
c       defined inter-weld positions, and can generate a joint interpolated
c       motion between two positions to check for interference.  Motion can
c       be forward or reverse, and the step size can be interactively changed.
c       The user can add new positions, remove existing positions or relocate
c       positions.

c       Subroutines called: editpa,posobs

c       Called from: main

c       Parameters:

c               lunit-  a logic unit number
c               changm- .true. if a modelling change has been made
c                       This forces a reconstruction of the scene

c       Variables:

c               add,rem,mov- .true. to add, remove or move a position
c               edit-   .true. to interactively modify the inter-weld path
c               forwrd,revers- forward or reverse motion
c               incrml- .true. to jump to the next inter-weld position
c               incr,decr- .true. to increase or decrease the step size
c               mincha- the minimum allowable changes in the joint values
c                       during a joint interpolated move
c               nobj-   number of inter-weld positions
c               posit-  array of pointers to inter-weld positions in file
c                       'ilocat'
c               start2- .true. at the start of a joint interpolated motion
c               step-   .true. for single step motion
c               values- robot and positioner joint values
c               vnext-  'values' for the next position
c
c       Saved variables:

c               icount- the inter-weld position from which the current joint
c                       interpolated motion starts
c               icoun2- the current number of steps along the joint interpolated
c                       motion
c               isteps- the size of an incremental move in number of steps
c               nmax-   the total number of steps from the initial position
c                       to the final position
c               scale-  a multiplier for the joint increments
c               vdiff-  the minimum joint increments

c*******************************************************************************

$include /usr/include/fdevice.h
$include /usr/include/fgl.h

        integer posit(10)
        integer nobjs,idseam
        character*20 ilocat
```

```
      real values(10,3),vnext(10,3),vdiff(10,3)
      real vdist
      integer*2 azim,inc,twist
      integer nrdof(3)
      logical start2
      logical incrml,changm
      real mincha(10,3)
      logical forwrd,revers
      logical edit,add,rem,mov
      logical incr,decr
      logical first
      logical step
      integer recl(20)

      save icount,icoun2,vdiff,nmax,scale,isteps,first

      common /start2/start2
      common /idseam/idseam
      common /idweld/idweld
      common /view/vdist,azim,inc,twist
      common /incrml/incrml
      common /nchain/nchain
      common /nrdof/nrdof
      common /ilocat/ilocat
      common /direct/forwrd,revers
      common /edit/edit,add,rem,mov
      common /values/values
      common /incr/incr
      common /decr/decr
      common /mincha/mincha
      common /step/step
      common /reclen/recl

c     Set the maximum number of positions (the same as the dimension of
c     'posit' )

      npmax=10

c     Initialize saved variables and show first robot position.

      if (start2) then
         isteps=8
         scale=isteps
         icount=1
         icoun2=0
         nmax=0
         start2=.false.

c        The selected path is the one following the picked weld rather than
c        preceeding it.

         if (forwrd) then
            idweld=idweld+1
         endif
         forwrd=.false.
         revers=.false.
```

```
c          Find the pointers to the positions.

           call posobs(lunit,idweld,posit,npmax,nobjs)

c          Read the first position.

           open(unit=lunit,file=ilocat,form='unformatted',access='direct',
     *          status='old',recl=recl(8))
           read(lunit,rec=posit(1)) ((values(j,i),j=1,nrdof(i)),i=1,nchain),
     *        time
           close(lunit)

           changm=.true.

        endif

c       Edit the inter-seam path.

        if (edit) then

c          In the path editing mode, interactively move the robot and positioner
c          and return the new joint values.

           call editpa(lunit,idweld,posit(icount),values,changm)


c       Step forward or backward along a joint interpolated path between two
c       given positions.


        elseif (forwrd.or.revers) then


c       Increase or decrease the step size by a factor of 2.

        if (icoun2.gt.0) then
           if (incr) then
              incr=.false.
              if (isteps.lt.nmax) then
                 scale=2.*scale
                 isteps=2*isteps
              endif
           elseif (decr) then
              decr=.false.
              if (isteps.gt.1) then
                 scale=scale/2.
                 isteps=isteps/2
              endif
           endif
        endif

c       At the start of the joint interpolated motion, read the joint values
c       of the current and next position, and calculate the appropriate
c       step sizes for the interpolated motion.
```

```
          if (icoun2.eq.0) then
             if (icount.eq.0) icount=1
             if (icount.ge.nobjs) icount=1
             open(unit=lunit,file=ilocat,form='unformatted',access='direct',
     *            status='old',recl=recl(8))

c            Read the two positions.

             read(lunit,rec=posit(icount)) (values(i,1),i=1,nrdof(1)),
     *          (values(i,2),i=1,nrdof(2)),time
             read(lunit,rec=posit(icount+1)) (vnext(i,1),i=1,nrdof(1)),
     *          (vnext(i,2),i=1,nrdof(2)),time
             close(lunit)

c            Calculate the number of interpolation steps to give the desired
c            maximum joint change.

             ntemp=0
             do 15 ind2=1,2
             do 10 ind=1,nrdof(ind2)
                vdiff(ind,ind2)=vnext(ind,ind2)-values(ind,ind2)
                numb=int(abs(vdiff(ind,ind2))/mincha(ind,ind2))
                if (numb.gt.ntemp) then
                   ntemp=numb
                endif
10           continue
15           continue

c            Change the number of steps to the next higher power of 2.

             nmax=1
100          continue
             if (nmax.ge.ntemp) then
                goto 200
             else
                nmax=nmax*2
                goto 100
             endif
200          continue

c            Calculate the corresponding joint changes per interpolation step.

             do 20 ind2=1,2
                do 25 ind=1,nrdof(ind2)
                   vdiff(ind,ind2)=vdiff(ind,ind2)/nmax
25              continue
20           continue

             changm=.true.

c       Increase or decrease the step size by a factor of 2.

          else

             if (incr) then
```

```
                 incr=.false.
                 if (isteps.lt.nmax) then
                    scale=2.*scale
                    isteps=2*isteps
                 endif
              elseif (decr) then
                 decr=.false.
                 if (isteps.gt.1) then
                    scale=scale/2.
                    isteps=isteps/2
                 endif
              endif

           endif


c          When the inter-weld path finishes, it is repeated from the start.
c          Disable the incremental motion so that the initial position is
c          shown.

           if (first) then
              first=.false.
           else

c          Step forward or backward along joint interpolated path.
c          If the current step is larger than the remaining motion, decrease
c          the step size and try again.

           if (forwrd) then
              icoun2=icoun2+isteps
              if (icoun2.ge.nmax+1) then
                 decr=.true.
                 icoun2=icoun2-isteps
              endif
           elseif (revers) then
              icoun2=icoun2-isteps
              if (icoun2.le.0) then
                 decr=.true.
                 icoun2=icoun2+isteps
              endif
           endif

c          Calculate new joint variables.

           if (.not.decr) then
              if (icoun2.gt.0) then
                 do 30 ind2=1,2
                    do 35 ind=1,nrdof(ind2)
                       if (forwrd) then
                          values(ind,ind2)=values(ind,ind2)+
     *                       scale*vdiff(ind,ind2)
                       elseif (revers) then
                          values(ind,ind2)=values(ind,ind2)-
     *                       scale*vdiff(ind,ind2)
                       endif
35                  continue
```

```
30                continue
             endif

             changm=.true.

c            If the final position is reached, repeat the motion from the
c            initial position.

             if (icoun2.eq.nmax) then
                icoun2=0
                first=.true.
             endif
          endif

c         If in single step mode, disable motion after one step.

          if (step) then
             forwrd=.false.
             revers=.false.
          endif

          endif
       endif

c      Step to the next given position, and show it.

       if (incrml) then
          forwrd=.false.
          revers=.false.
          icount=icount+1
          if (icount.gt.nobjs) icount=1
          open(unit=lunit,file=ilocat,form='unformatted',access='direct',
     *         status='old',recl=recl(8))

          read(lunit,rec=posit(icount)) (values(i,1),i=1,nrdof(1)),
     *          (values(i,2),i=1,nrdof(2)),time
          close(lunit)
          changm=.true.
          icoun2=0
          incrml=.false.
       endif

       return
       end
```

```
        subroutine disjnt(parts,result)

c       Test if the projections of two polyhedral parts overlap on the screen.

c       CALLED FROM:
c               ohtest

c       SUBROUTINES CALLED:
C               cvhull

c       PARAMETERS:
c               parts(2)-       indices of two parts to be compared
c               result-         .true. if the parts are disjoint in screen space

c       Variables:
c               nvert-  number of vertices on the convex hulls of the
c                       projections
c               slistl,slist2-  lists of vertices on the convex hulls
c               slist-  combination of 'slistl','slist2'
c******************************************************************************


        integer slistl(2,100),slist2(2,100),slist(2,2,100)

        integer parts(2)
        integer nvert(2)
        logical result

        toler=1.5


c       Find the convex hulls of the projections of the two parts into screen
c       space.

        call cvhull(parts(1),slistl,nvert(1))
        call cvhull(parts(2),slist2,nvert(2))
        do 12 i=1,nvert(1)
           do 13 j=1,2
              slist(1,j,i)=slistl(j,i)
13         continue
12      continue
        do 16 i=1,nvert(2)
           do 17 j=1,2
              slist(2,j,i)=slist2(j,i)
17         continue
16      continue


c       The polygon screen projections are disjoint if all vertices of
c       one hull lie "outside" of any edge of the other hull.

        result=.false.

c       test each hull against the other
```

```
        do 15 ipoly=1,2
        if (.not.result) then
           do 20 iverts=1,nvert(ipoly)
              if (.not.result) then
                 inext=iverts+1
                 if (inext.gt.nvert(ipoly)) inext=1

c       find the line equation coefficients of an edge

                 dx=float(slist(ipoly,1,inext)-slist(ipoly,1,iverts))
                 dy=float(slist(ipoly,2,inext)-slist(ipoly,2,iverts))

c                Test if the edge endpoints are disjoint.

                 dlen=sqrt(dx**2+dy**2)
                 if (dlen.ne.0.) then

c                   Swap x, y to get the minimum slope and to avoid singularit

                    if (dx.gt.dy) then
                       a=dy/dx
                       ind1=1
                       ind2=2
                    else
                       a=dx/dy
                       ind1=2
                       ind2=1
                    endif

c                   Find the intercept for the test edge, and define it as
c                   the origin.

                    test=-a*slist(ipoly,ind1,iverts)+slist(ipoly,ind2,iverts)

c                   Find the intercept of a parallel line containing another
c                   point on the hull. The sign indicates the orientation of
c                   the edge.

                    test1=0.
                    do 88 ivrt2=1,nvert(ipoly)
                       if (abs(test1).le..0001) then
                          test1=-a*slist(ipoly,ind1,ivrt2)+
     *                         slist(ipoly,ind2,ivrt2)-test
                       endif
88                  continue

c       Plug all the vertices of the other face into the line equation.
c       If they all lie "outside" the edge,
c       their intercepts will all have the opposite sign from 'test1', and
c       the faces are disjoint.

                    iother=2
                    if(ipoly.eq.2) iother=1
                    result=.true.
                    do 25 ivert2=1,nvert(iother)
```

```
                       if (result) then
                          test2=-a*slist(iother,ind1,ivert2)+
     *                         slist(iother,ind2,ivert2)-test
                          if(test1*test2.gt.0.and.abs(test2).gt.toler)
     *                       result=.false.
                       endif
25                  continue
                 endif
              endif
20         continue
           endif
15      continue

        return
        end
```

```
      subroutine displl

c     Display important information in the menu area of the screen.

c     Subroutines called: setmen

c     Called from: intest,main

c     Variables:
c            defwel- .true. when picking seams to weld
c            edit-   .true. when editing inter-weld path
c            intest- .true. during weld trajectory generation
c            intset- .true. to record interfering paths during replay
c            makep-  .true. when processing a selected inter-weld path
c            mkpath- .true. when entering inter-weld path mode
c            movtab- initial and final table positions
c            mvtabl- .true. for positioning of movable table
c            mindis- the minimum separating distance between parts
c            nrdof-  the degrees of freedom of the robot and positioner
c            replay- .true. during replay of robot program
c            shdist- .true. when calculating exact separating distances
c            step-   .true. for single step motion mode
c            tesint- .true. when testing for interference
c            time1-  elapsed time of current weld
c            time2-  total elapsed time of all welds
c            time3-  elapsed time of inter-weld path
c            time4-  total inter-weld elapsed time
c            time5-  total elapsed program time
c            values- the robot and positioner joint values
c
c

c***********************************************************************

      real mindis
      integer nrdof(3)
      real values(10,3)
      character*10 string
      character*5 shstri
      integer left,top,idely
      real movtab(10,2),tstep(10)
      logical keyon,replay,shdist,pickfl,solid,shrobt
      logical mvtabl,tinit,tfinal
      logical edit,add,rem,mov
      logical defwel,makep
      logical menul
      logical mkpath
      logical tesint
      logical intset
      logical step
      logical intest

      common /nrdof/nrdof
      common /mindis/mindis
      common /values/values
      common /flagsl/keyon,replay,shdist,pickfl,solid,shrobt
      common /flags2/mvtabl,tinit,tfinal
      common /movtab/movtab,tstep
      common /exprt/volt,wsize,feed,gfr,walpha,beta,stkout
      common /currnt/currnt
      common /edit/edit,add,rem,mov
      common /defwel/defwel
      common /makep/makep
      common /mkpath/mkpath
      common /menu1/menu1
      common /tesint/tesint
      common /intset/intset
      common /step/step
      common /intest/intest
      common /time/time1,time2,time3,time4,time5

      left=760
      top=750
      idely=15

      call pushma
      call pushat
      call pushvi

      call color(i2(11))
      call setmen

c     Show minimum separating distance.

      call cmov2i(left,top)
      if (tesint.or.shdist.and..not.intest) then
         call charst('Min. Distance  ')
         write(unit=string,fmt='(f10.4)') mindis
         call charst(string)
      else
         call charst(' ')
      endif

c     Show current table position.

      top=top-3*idely
      call cmov2i(left,top)
      call charst('Current table position')
      top=top-2*idely
      do 10 i=1,nrdof(2)
         call cmov2i(left,top)
         write(unit=shstri,fmt='(i2)') i
         call charst(shstri)
         write(unit=string,fmt='(f8.2)') values(i,2)
         call pushat
         call color(i2(15))
         call charst(string)
         call popatt
         top=top-idely
10    continue

c     Show initial and final table positions.
```

```
      if (mvtabl) then
         top=top-2*idely
         call cmov2i(left,top)
         call charst('Initial and final')
         top=top-idely
         call cmov2i(left,top)
         call charst('positions')
         top=top-2*idely
         do 30 i=1,nrdof(2)
            call cmov2i(left,top)
            write(unit=shstri,fmt='(i2)') i
            call charst(shstri)
            write(unit=string,fmt='(f8.2)') movtab(i,1)
            call charst(string)
            write(unit=string,fmt='(f8.2)') movtab(i,2)
            call charst(string)
            top=top-idely
30       continue
      endif

c     Show current robot joint values.

      top=top-2*idely
      call cmov2i(left,top)
      call charst('Robot Joint Variables')
      top=top-2*idely
      do 15 i=1,nrdof(1)
         call cmov2i(left,top)
         write(unit=shstri,fmt='(i2)') i
         call charst(shstri)
         write(unit=string,fmt='(f8.2)') values(i,1)
         call pushat
         call color(i2(15))
         call charst(string)
         call popatt
         top=top-idely
15    continue

      if (.not.replay.and..not.intest) then

c     Show global button functions.

      top=top-2*idely
      call cmov2i(left,top)
      call charst('User options:')
      top=top-2*idely
      call cmov2i(left,top)
      call charst('PF1: Quit')

      top=top-idely
      call cmov2i(left,top)
      call charst('PF2: Change mouse buttons')

      top=top-idely
      call cmov2i(left,top)
```

```
        call charst('PF3: Reset')

        top=top-idely
        call cmov2i(left,top)
        call charst('PF4: Replay program')

        top=top-idely
        call cmov2i(left,top)
        call charst('ENTER 1: Solid/Wireframe')

        top=top-idely
        call cmov2i(left,top)
        call charst('ENTER 2: Highlight seams-  ')

        top=top-idely
        call cmov2i(left,top)
        call charst('ENTER 3: Show distances-  ')
        if (shdist) then
           call charst('yes')
        else
           call charst('no')
        endif

        top=top-idely
        call cmov2i(left,top)
        call charst('ENTER 4: Test interference- ')
        if (tesint) then
           call charst('yes')
        else
           call charst('no')
        endif

        top=top-idely
        call cmov2i(left,top)
        call charst('ENTER 5: Interfering paths- ')
        if (intset) then
           call charst('yes')
        else
           call charst('no')
        endif

        if (menu1) then
c
           Show choice of modes.

           top=top-idely
           call cmov2i(left,top)
           call charst('1: Define seams')

           top=top-idely
           call cmov2i(left,top)
           call charst('2: Define welds')

           top=top-idely
           call cmov2i(left,top)
           call charst('3: Inter-weld path')
        endif

        if (defwel) then

c       Show options in weld definition mode.

        top=top-2*idely
        call cmov2i(left,top)
        call charst('0: Quit')

        top=top-idely
        call cmov2i(left,top)
        call charst('1: Remove seam')

        top=top-idely
        call cmov2i(left,top)
        call charst('2: Reverse direction')

        top=top-idely
        call cmov2i(left,top)
        call charst('3: Unweld seam')

        top=top-idely
        call cmov2i(left,top)
        call charst('4: Show order')

        top=top-idely
        call cmov2i(left,top)
        call charst('7: Move table')

        if (mvtabl) then

        top=top-idely
        call cmov2i(left,top)
        call charst('8:  Initial table position')

        top=top-idely
        call cmov2i(left,top)
        call charst('9:  Final table position')

        endif
        endif

        if ((makep.or.mkpath).and..not.edit) then

c       Show options in inter-weld path mode.

        top=top-2*idely
        call cmov2i(left,top)
        call charst('0: Quit')

        top=top-idely
        call cmov2i(left,top)
        call charst('1: Incr. position')
        top=top-idely
        call cmov2i(left,top)
        call charst('3: Edit path')

        top=top-idely
        call cmov2i(left,top)
        call charst('4: Double step size')

        top=top-idely
        call cmov2i(left,top)
        call charst('5: Halve step size')

        top=top-idely
        call cmov2i(left,top)
        call charst('6: Single step mode- ')
        if (step) then
           call charst('yes')
        else
           call charst('no')
        endif

        top=top-idely
        call cmov2i(left,top)
        call charst('7: Next interfering path')

        top=top-idely
        call cmov2i(left,top)
        call charst('->: Move forward')

        top=top-idely
        call cmov2i(left,top)
        call charst('<-: Move backward')

        endif

        if (edit) then

c       Show options in inter-path editing mode.

        top=top-2*idely
        call cmov2i(left,top)
        call charst('0: Quit')

        top=top-idely
        call cmov2i(left,top)
        call charst('R: Move robot joints')

        top=top-idely
        call cmov2i(left,top)
        call charst('T: Move positioner joints')

        top=top-idely
        call cmov2i(left,top)
        call charst('->: Increase joint value')

        top=top-idely
        call cmov2i(left,top)
        call charst('<-: Decrease joint value')

        top=top-idely
        call cmov2i(left,top)
        call charst('1: Add position')

        top=top-idely
        call cmov2i(left,top)
        call charst('2: Remove position')

        top=top-idely
        call cmov2i(left,top)
        call charst('3: Move position')

        top=top-idely
        call cmov2i(left,top)
        call charst('6: Single step mode- ')
        if (step) then
           call charst('yes')
        else
           call charst('no')
        endif

        endif


        elseif (replay) then

c          Show welding process variables in replay mode.

           top=top-3*idely
           call cmov2i(left,top)
           call charst('Wire size-  ')
           write(unit=string,fmt='(f8.3)') wsize
           call charst(string)

           top=top-2*idely
           call cmov2i(left,top)
           call charst('Voltage-    ')
           write(unit=string,fmt='(f8.2)') volt
           call charst(string)

           top=top-idely
           call cmov2i(left,top)
           call charst('Feed rate-  ')
           write(unit=string,fmt='(f8.2)') feed
           call charst(string)

           top=top-idely
           call cmov2i(left,top)
           call charst('Gas flow-   ')
           write(unit=string,fmt='(f8.2)') gfr
           call charst(string)

           top=top-idely
```

```
            call cmov2i(left,top)
            call charst('Current-     ')
            write(unit=string,fmt='(f8.2)') currnt
            call charst(string)

            top=top-2*idely
            call cmov2i(left,top)
            call charst('Elapsed Welding Time')

            top=top-idely
            call cmov2i(left,top)
            call charst('Current-     ')
            write(unit=string,fmt='(f8.2)') timel
            call charst(string)

            top=top-idely
            call cmov2i(left,top)
            call charst('Total-       ')
            write(unit=string,fmt='(f8.2)') time2
            call charst(string)

            top=top-2*idely
            call cmov2i(left,top)
            call charst('Elapsed  Inter-weld Time')

            top=top-idely
            call cmov2i(left,top)
            call charst('Current-     ')
            write(unit=string,fmt='(f8.2)') time3
            call charst(string)

            top=top-idely
            call cmov2i(left,top)
            call charst('Total-       ')
            write(unit=string,fmt='(f8.2)') time4
            call charst(string)

            top=top-2*idely
            call cmov2i(left,top)
            call charst('Elapsed Program Time')

            top=top-idely
            call cmov2i(left,top)
            call charst('Total-       ')
            write(unit=string,fmt='(f8.2)') time5
            call charst(string)

c           Show user options.

            top=top-2*idely
            call cmov2i(left,top)
            call charst('PF4- STOP REPLAY')

            top=top-2*idely
            call cmov2i(left,top)
            call charst('Pad 1- Skip section')
```

```
          top=top-idely
          call cmov2i(left,top)
          call charst('Pad 4- Double step size')

          top=top-idely
          call cmov2i(left,top)
          call charst('Pad 5- Halve step size')

       elseif (intest) then

c         Show exit button in weld generation mode.

          top=top-2*idely
          call cmov2i(left,top)
          call charst('Pad 0- Quit')

       endif

       call gflush
       call popvie
       call popatt
       call popmat
       return
       end
```

```
      subroutine drpart(plist,number)

c     Draw a set of convex polyhedral parts as an ordered set of shaded,
c     filled polygons. Faces may have up to 20 vertices.

c     Called from: drwkst

c     Subroutines called:  drpoly, facvrt, fplist, getcrd, getnml,
c                          getvrt, vrtlst

c     Parameters:      plist- an ordered list of the convex polyhedra to be
c                             drawn
c                      number- the number of polyhedra in the list

c*****************************************************************************

      integer plist(*),number
      integer fstart,fend,vstart,vend
      real coords(3),coord2(3),normal(3)
      real dotprd,length
      integer*2 shade
      real array3(3,3),array4(3,4),array5(3,5),array6(3,6),array7(3,7)
      real array8(3,8),array9(3,9),aray10(3,10),aray11(3,11),aray12(3,12)
      real aray13(3,13),aray14(3,14),aray15(3,15),aray16(3,16),aray17(3,17)
      real aray18(3,18),aray19(3,19),aray20(3,20)


c     Draw each polyhedron in the list, in order.
c     For each face, calculate the face orientation relative to the view
c     vector. Discard self-hidden faces and calculate a shade for the visible
c     faces.

      do 10 ipart=1,number
         call fplist(plist(ipart),istart,iend)
         do 20 iface=istart,iend
            call facvrt(iface,coords)
            call getnml(iface,normal)
            dotprd=0.
            length=0.
            do 30 i=1,3
               dotprd=dotprd+coords(i)*normal(i)
               length=length+coords(i)**2
30          continue
            dotprd=dotprd/sqrt(length)

c     Output the visible faces as filled polygons.

            if (dotprd.lt.0.) then

c              Calculate the shade from the dot product.

               shade=int(-11*dotprd)
               call color(shade)
```

```
                  call vrtlst(iface,vstart,vend)
                  nvert=vend-vstart+1

                  if (nvert.eq.3) then
                     call drpoly(nvert,vstart,array3)
                  elseif (nvert.eq.4) then
                     call drpoly(nvert,vstart,array4)
                  elseif (nvert.eq.5) then
                     call drpoly(nvert,vstart,array5)
                  elseif (nvert.eq.6) then
                     call drpoly(nvert,vstart,array6)
                  elseif (nvert.eq.7) then
                     call drpoly(nvert,vstart,array7)
                  elseif (nvert.eq.8) then
                     call drpoly(nvert,vstart,array8)
                  elseif (nvert.eq.9) then
                     call drpoly(nvert,vstart,array9)
                  elseif (nvert.eq.10) then
                     call drpoly(nvert,vstart,aray10)
                  elseif (nvert.eq.11) then
                     call drpoly(nvert,vstart,aray11)
                  elseif (nvert.eq.12) then
                     call drpoly(nvert,vstart,aray12)
                  elseif (nvert.eq.13) then
                     call drpoly(nvert,vstart,aray13)
                  elseif (nvert.eq.14) then
                     call drpoly(nvert,vstart,aray14)
                  elseif (nvert.eq.15) then
                     call drpoly(nvert,vstart,aray15)
                  elseif (nvert.eq.16) then
                     call drpoly(nvert,vstart,aray16)
                  elseif (nvert.eq.17) then
                     call drpoly(nvert,vstart,aray17)
                  elseif (nvert.eq.18) then
                     call drpoly(nvert,vstart,aray18)
                  elseif (nvert.eq.19) then
                     call drpoly(nvert,vstart,aray19)
                  elseif (nvert.eq.20) then
                     call drpoly(nvert,vstart,aray20)
                  else
                     write(*,*) 'error- face contains illegal number of vertices'
                  endif


c        Draw a boundary around each filled polygon.

                  call color(i2(15))
                  call move(coords(1),coords(2),coords(3))
                  do 40 ind=vstart,vend
                     call getvrt(ind,ipoint)
                     call getcrd(ipoint,coord2)
                     call draw(coord2(1),coord2(2),coord2(3))
40                continue
                  call draw(coords(1),coords(2),coords(3))
              endif
```

```
20          continue
10      continue


        return
        end
```

```
      subroutine drpoly(numb,index,array)
c     This subroutine maps the face vertices into an array and then draws
c     a filled polygon.
c     CALLED FROM:    drpart
c     SUBROUTINES CALLED: getcrd, getvrt
c
c     IRIS GRAPHICS CALLS:
c             polf
c     PARAMETERS:
c             numb-   the number of vertices on the face
c             index-  the location of the first index in the vertex list
c             array-  the array of vertices
c*****************************************************************************
      integer numb,index
      real array(3,numb),temp(3)

      iarray=1
      do 35 icount=index,index+numb-1
         call getvrt(icount,ivert)
         call getcrd(ivert,temp)
         do 36 kk=1,3
            array(kk,iarray)=temp(kk)
36       continue
         iarray=iarray+1
35    continue
      call polf(numb,array)
      return
      end
      subroutine drscrn

c     Define a window and draw world coordinate axes.

c     Subroutines called: none

c     Graphics calls: cmov,move,draw,pushma,popmat,window,polarv,charst

c     Called from: intest,main
c******************************************************************************

      real winsiz(6)
      real vdist
      integer*2 azim,inc,twist

      common /winsiz/winsiz
      common /view/vdist,azim,inc,twist


      call pushma
      call window(winsiz(1),winsiz(2),winsiz(3),winsiz(4),winsiz(5),winsiz(6))
      call polarv(vdist,azim,inc,twist)
      call move(0.,0.,0.)
      call draw(800.,0.,0.)
      call move(0.,0.,0.)
      call draw(0.,800.,0.)
      call move(0.,0.,0.)
      call draw(0.,0.,800.)
      call cmov(830.,0.,0.)
      call charst('X')
      call cmov(0.,830.,0.)
      call charst('Y')
      call cmov(0.,0.,830.)
      call charst('Z')
      call popmat


      return
      end
      SUBROUTINE DRSEAM(seamob,boldob,endobj)
C
C     THIS SUBROUTINE DEFINES GRAPHICAL OBJECTS REPRESENTING THE SEAMS.

c     Parameters:      seamob- object id of a seam.
c                      boldob- object id of a highlighted seam.
c                      endobj- object id of the set of seam boundary glyphs.

c     Subroutines called: none

c     Graphics calls: closeo,makeob,maketa,move,pnt

c     Called from: main

c     Common blocks modified: none

c     Files read: cadlst,procad

c     Files written: none

c     Variables:

c             nseams- the number of seam segments in the CAD seam descriptio.
c             r-      coordinates of a seam point in the workpiece frame
c             t-      seam tangent vector
c             a-      seam normal vector
c             b-      seam lateral vector
C
C*****************************************************************************
C
C
      character*20 cadlst
      character*20 procad
      REAL WINSIZ(6)
      REAL R(3),A(3)
      REAL T(3),B(3)
      REAL STEMP
      integer seamob,boldob,endobj
      integer recl(20)
C
      COMMON /WINSIZ/WINSIZ
      common /cadlst/cadlst
      common /procad/procad
      common /nseams/nseams
      common /reclen/recl

      delt=5.
C
C
C*****  OPEN THE LIST OF SEAM DATA FILES.
C
      open(unit=1,file=cadlst,form='unformatted',access='direct',
     *     status='old',recl=recl(2))
      open(unit=2,file=procad,form='unformatted',access='direct',
     *     status='old',recl=recl(4))
C
```

```
c       Create reference seam object. This object is for picking seam points.

        call makeob(seamob)
C
C
C*****  REPEAT FOR ALL SEAMS
C
        DO 100 ISEAM=1,NSEAMS
C
           read(1,rec=iseam) nstart,npoint

C*****     CREATE A SEGMENT I
C
           call maketa(iseam)
           call move(0.,0.,0.)
C
C
C*****        READ SEAM DATA FOR CURRENT POINT

           do 40 ind=nstart,nstart+npoint-1
              read(2,rec=ind) stemp,(r(i),i=1,3),(t(j),j=1,3),(a(k),k=1,3),
     *                    (b(l),l=1,3)
C
              call pnt(r(1),r(2),r(3))
C
40         continue
C
100     CONTINUE
C
        call closeo

c       Create a bold seam object for easy viewing.

        call makeob(boldob)
C
C
C*****  REPEAT FOR ALL SEAMS
C
        DO 300 ISEAM=1,NSEAMS
C
           read(1,rec=iseam) nstart,npoint

C
C*****        READ SEAM DATA FOR CURRENT POINT

           do 60 ind=nstart,nstart+npoint-1
              read(2,rec=ind) stemp,(r(i),i=1,3),(t(j),j=1,3),(a(k),k=1,3),
     *                    (b(l),l=1,3)
C

              if (ind.eq.nstart) then
                 call move(r(1),r(2),r(3))
              else
                 call draw(r(1),r(2),r(3))
              endif
```

```
                call move(r(1)-delt*b(1),r(2)-delt*b(2),r(3)-delt*b(3))
                call draw(r(1)+delt*b(1),r(2)+delt*b(2),r(3)+delt*b(3))
                call move(r(1),r(2),r(3))
C
60           continue

300       continue

          call closeo
C

c         Create endpoint glyphs to separate the segments of the reference seam.

          call makeob(endobj)

          do 200 iseam=1,nseams

             read(1,rec=iseam) nstart,npoint

             read(2,rec=nstart) stemp,(r(i),i=1,3),(t(j),j=1,3),(a(k),k=1,3),
     *                          (b(l),l=1,3)

             call move(r(1)-delt*b(1),r(2)-delt*b(2),r(3)-delt*b(3))
             call draw(r(1)+delt*b(1),r(2)+delt*b(2),r(3)+delt*b(3))
             call move(r(1)-delt*a(1),r(2)-delt*a(2),r(3)-delt*a(3))
             call draw(r(1)+delt*a(1),r(2)+delt*a(2),r(3)+delt*a(3))

             read(2,rec=nstart+npoint-1) stemp,(r(i),i=1,3),(t(j),j=1,3),
     *                                   (a(k),k=1,3),(b(l),l=1,3)

             call move(r(1)-delt*b(1),r(2)-delt*b(2),r(3)-delt*b(3))
             call draw(r(1)+delt*b(1),r(2)+delt*b(2),r(3)+delt*b(3))
             call move(r(1)-delt*a(1),r(2)-delt*a(2),r(3)-delt*a(3))
             call draw(r(1)+delt*a(1),r(2)+delt*a(2),r(3)+delt*a(3))

200       continue

          call closeo

          close(1)
          close(2)
C
          call gflush
C
          RETURN
          END
```

```
      SUBROUTINE drsequ(files,nfile,iobj)
C
C     THIS SUBROUTINE DEFINES GRAPHICAL OBJECTS REPRESENTING THE SEAMS.
C
c     Parameters:     files-  an array of seam segments
c                             files(1,*)- the reference seam id
c                             files(2,*)- the starting point
c                             files(3,*)- the end point
c
c                     nfile-  the number of segments
c                     iobj-   a unique graphical object number
c                             corresponding to the
c                             defined seam
c
c     Subroutines called: none
c
c     Graphics calls: yes
c
c     Called from: main,mksequ,revdir
c
c     Parameters modified:  iobj
c
c     Common blocks modified: none
c
c     Files read: 'procad'
c
c     Files written: none
c
c     Variables:
c
c               R-      seam position vector
c               T-      seam tangent vector
c               A-      seam normal vector
c               B-      seam lateral vector
c               p1,p2,p3,p4-  temporary points
C
C*****************************************************************************
C
$include /usr/include/fgl.h
C
      character*20 procad
      REAL R(3),A(3)
      REAL T(3),B(3)
      integer files(3,*),nfile,iobj
      real p1(3),p2(3),p3(3),p4(3)
      integer recl(20)
C
      common /procad/procad
      common /reclen/recl


      if (nfile.gt.0) then
      del=10.
C
C
C*****  OPEN THE SEAM DATA FILE.
```

```
C
        open(unit=2,file=procad,form='unformatted',access='direct',
     *       status='old',recl=recl(4))
C

        iobj=genobj()
        call makeob(iobj)
C
C
C*****  REPEAT FOR ALL SEAMS
C
        do 100 ind2=1,nfile
           istart=files(2,ind2)
           iend=files(3,ind2)
           if (iend.lt.istart) then
              incr=-1
           else
              incr=1
           endif
           do 40 ind=istart,iend,incr

c             Read seam data.

              read(2,rec=ind) stemp,(r(i),i=1,3),(t(j),j=1,3),(a(k),k=1,3),
     *                    (b(l),l=1,3)
C
              if (ind2.eq.1.and.ind.eq.istart) then

c                Draw a cross at the start of the seam.

                 call move(r(1)+del*a(1),r(2)+del*a(2),r(3)+del*a(3))
                 call draw(r(1)-del*a(1),r(2)-del*a(2),r(3)-del*a(3))
                 call move(r(1)+del*b(1),r(2)+del*b(2),r(3)+del*b(3))
                 call draw(r(1)-del*b(1),r(2)-del*b(2),r(3)-del*b(3))

                 call move(r(1),r(2),r(3))
              else
                 call draw(r(1),r(2),r(3))
              endif
C
40         continue

C
100     CONTINUE

c       Draw an arrow at the end of the seam.

        do 200 i=1,3
           if (incr.eq.-1) t(i)=-t(i)
           p1(i)=r(i)-del*(2.*t(i)+a(i))
           p2(i)=r(i)-del*(2.*t(i)-a(i))
           p3(i)=r(i)-del*(2.*t(i)+b(i))
           p4(i)=r(i)-del*(2.*t(i)-b(i))
200     continue

        call draw(p1(1),p1(2),p1(3))
```

```
      call move(r(1),r(2),R(3))
      call draw(p2(1),p2(2),p2(3))
      call move(r(1),r(2),r(3))
      call draw(p3(1),p3(2),p3(3))
      call move(r(1),r(2),r(3))
      call draw(p4(1),p4(2),p4(3))
      call draw(p1(1),p1(2),p1(3))
      call draw(p3(1),p3(2),p3(3))
      call draw(p2(1),p2(2),p2(3))
      call draw(p4(1),p4(2),p4(3))


c
      call closeo
      close(2)
c
      call gflush
      endif
c
      RETURN
      END
```

```
        subroutine drtext(strin1,strin2)

c       Write a string of graphics text into an allocated area on the screen

c       Graphics calls: yes

c       Called from: editpa,genpat,input,mkseam,smpick

c******************************************************************************


        character*(*) strin1
        character*(*) strin2
        integer*2 black,white
        real ident(4,4)

        common /ident/ident


        ileft=0
        iright=700
        ibotto=751
        itop=767
        ix=10
        iy=753
        white=11
        black=0

        call pushat
        call pushma
        call pushvi
        call frontb(.true.)
        call loadma(ident)

c       Define a text area.

        call ortho2(0.,767.,0.,767.)
        call viewpo(i2(0),i2(767),i2(0),i2(767))
        call color(black)
        call rectfi(ileft,ibotto,iright,itop)
        call color(white)
        call recti(ileft,ibotto,iright,itop)

c       Output the text string.

        call cmov2i(ix,iy)
        call charst(strin1)
        call charst(strin2)

        call gflush
        call frontb(.false.)
        call popvie
        call popmat
        call popatt

        return
```

```
      end
```

```
        subroutine drwire(itype,plist,number)

c       Draw a wireframe depiction of a set of convex polyhedral objects.

c       Subroutines called:  facvrt, fplist, getcrd, getnml, getvrt, vrtlst

c       Graphics calls: draw,move

c       Called from: drwkst,intest,main

c       Parameters:  itype- 0- all edges are shown
c                           1- self-hidden faces are not shown.
c                              (coords. must be in viewing coords.)
c                    plist- list of parts to be shown
c                    number- the number of parts

c       Variables:
c               fstart,fend- starting and ending faces
c               vstart,vend- starting and ending vertices
c               coords,coord2- vertex coordinates
c               normal- face normal
c               array*- arrays containing polygons for output
c********************************************************************************


        integer plist(*),number
        integer fstart,fend,vstart,vend
        real coords(3),coord2(3),normal(3)
        real dotprd,length
        real array3(3,3),array4(3,4),array5(3,5),array6(3,6),array7(3,7)
        real array8(3,8),array9(3,9),aray10(3,10)

        do 10 ipart=1,number
           call fplist(plist(ipart),istart,iend)
           do 20 iface=istart,iend
              call facvrt(iface,coords)
              call getnml(iface,normal)

c             Test if a face is visible by taking the dot product of its
c             reference point and normal.  This only works if the points are
c             in viewing coordinates.

              if (itype.eq.1) then
                 dotprd=0.
                 length=0.
                 do 30 i=1,3
                    dotprd=dotprd+coords(i)*normal(i)
                    length=length+coords(i)**2
30               continue
                 dotprd=dotprd/sqrt(length)
              else
                 dotprd=-1.
              endif

              if (dotprd.lt.0.) then
```

```
                    call vrtlst(iface,vstart,vend)

c        Draw the boundary of each visible polygon.

                    call move(coords(1),coords(2),coords(3))
                    do 40 ind=vstart,vend
                       call getvrt(ind,ipoint)
                       call getcrd(ipoint,coord2)
                       call draw(coord2(1),coord2(2),coord2(3))
40                  continue
                    call draw(coords(1),coords(2),coords(3))
                 endif

20            continue
10       continue

         return
         end
```

```
      subroutine drwkst(part,number,shdist,changv,changm,solid)

c       Draw the robot, workpiece and table, showing seams and welds.

c       Parameters:     part-   an array of part ids.
c                       number- the number of parts.
c                       shdist- .true. if distances are to be calculated and
c                               shown
c                       changv- .true. if a viewing change has occurred
c                       changm- .true. if a modelling change has occurred
c                       solid-  .true. if a solid shaded depiction is desired.
c                               .false. if a wire frame depiction is desired.

c       Subroutines called:  drpart,drwire,plist,psort

c       Graphics calls: callob,color,gflush,multma,popmat,popatt,popvie,
c                       pushma,pushatt,pushvie,polarv,ringbe

c       Called from: main

c       Common blocks modified: none

c       Variables:
c               boldsm- .true. to highlight the CAD seam
c               boldob- graphic object for a highlighted CAD seam
c               colist- list of interfering pairs of parts
c               endobj- id of graphical object for seam boundaries
c               ftran1,ftran2-  transformations from the world frame to
c                               the torch and positioner frames
c               grobjs- list of graphical objects for user defined seams
c               icolst- number of interfering parts
c               intflg- .true. if interference occurs
c               ipick-  list of welded seams
c               icount- number of welded seams
c               mindis- the minimum separating distance
c               nseq-   number of user defined seams
c               seamob- id of CAD seam graphical object
c               shdist- .true. for calculation of exact separating distances
c               oparts- parts list ordered for hidden surface removal
c               shord-  .true. to show the order of the welds
c               tesint- .true. to test for interference
c               values- the robot and positioner joint values
c
c       Saved variables:
c               iseam-  weld id - this is incremented to show weld order

c***************************************************************************


$include /usr/include/fgl.h
$include /usr/include/fdevice.h

      integer colist(100),icolst
      integer nseq,seamob,boldob,endobj,grobjs(100)
      integer oparts(100)
      integer part(*),number
```

```
        logical shdist,changv,changm,solid
        real vdist,mindis
        integer*2 azim,inc,twist
        real values(10,3)
        real ftran1(4,4),ftran2(4,4)
        integer icount,ipick(100)
        logical shord
        logical tesint
        logical intflg
        logical boldsm

        save iseam

        common /view/vdist,azim,inc,twist
        common /values/values
        common /mindis/mindis
        common /ftrans/ftran1,ftran2
        common /nseq/nseq
        common /grobjs/grobjs
        common /seamob/seamob,boldob,endobj
        common /ipick/icount,ipick
        common /shord/shord
        common /tesint/tesint
        common /intflg/intflg
        common /boldsm/boldsm
c       Draw the robot, workpiece and table.

        call pushma
        call pushat
        call pushvi

c       If a modelling change occurs, the separating planes must be recalculated
c       to update the exact separating distance, for hidden surface removal and
c       for interference detection.

        if (changm) then
           if (shdist) then
              call plist(1,part,number,colist,icolst,mindis)
           elseif (solid.or.tesint) then
              call plist(0,part,number,colist,icolst,mindis)
           endif
        endif

c       For hidden surface removal for disjoint parts, recalculate order if the
c       scene has changed and draw the parts.

        if (solid.and.mindis.gt.0.) then
           if (changv.or.changm) then
              call pushvi
              call psort(part,number,oparts)
              call popvie
           endif
           call pushat
           call drpart(oparts,number)
           call gflush
           call popatt
        else

c          If interference has been detected, change the colour and output
c          a wireframe depiction.

           if (mindis.lt.0.and.tesint) then
              intflg=.true.
              call color(i2(12))
           else
              call color(i2(15))
           endif
           call pushma

c          In solid mode, the vertices are in viewing coordinates for easy
c          hidden surface removal.  In wire frame mode, the vertices are
c          in world coordinates for speed.

           if (.not.solid) call polarv(vdist,azim,inc,twist)
           call drwire(0,part,number)

c          Draw interfering parts in another colour.

           if (intflg) then
              call color(i2(13))
              call drwire(0,colist,icolst)
           endif
           call gflush
           call popmat
        endif

c       Draw the seams and completed welds. The graphic objects must be
c       transformed into workpiece coordinates.

        call polarv(vdist,azim,inc,twist)
        call multma(ftran2)

        call color(i2(12))
        if (boldsm) then
           call callob(boldob)
        else
           call callob(seamob)
        endif
        call color(i2(13))
        call callob(endobj)
        call color(i2(14))
        do 10 i=1,nseq
           call callob(grobjs(i))
10      continue

        call color(i2(13))

c       Show the welds in order.

        if (shord) then
```

```
            iseam=iseam+1
            if (iseam.gt.icount) iseam=0
         else
            iseam=icount
         endif

         do 15 i=1,iseam
            call callob(grobjs(ipick(i)))
15       continue

         call gflush

         call popatt
         call popmat
         call popvie

         return
         end
```

```
        subroutine edges(coord1,coord2,fcoord,fnumb,pnorm,pvert,dmin)

c       Find the shortest vector connecting an edge of one object to an edge
c       of the closest face of another object.

c       SUBROUTINES CALLED:
c               etest

c       Called from: finbes

c       PARAMETERS:
c               coord1- the first edge endpoint
c               coord2- the second edge endpoint
c               fcoord- the set of face vertices
c               fnumb-  the number of face vertices
c               pvert-  the starting point of the connecting vector
c               pnorm-  the connecting vector
c               dmin-   the minimum distance between the two objects
c                       (the length of the connecting vector)

c       Variables:
c               dist-   minimum distance between two test edges
c               v1,v2-  endpoints of two test edges
c               vert1-  starting point of the shortest connecting vector between
c                       two test edges
c               vector- the shortest connecting vector between two test edges

c******************************************************************************

        integer type,fnumb
        real coord1(3),coord2(3),pnorm(3),dist
        real pvert(3),fcoord(3,*),vector(3)
        real v1(3,2),v2(3,2),vert1(3)
        logical flag

        toler=.01


             do 25 i=1,3
                v1(i,2)=coord1(i)
25           continue
                do 30 i=1,3
                   v2(i,2)=coord2(i)
30              continue

c       Test the first edge against each edge of the closest face.

                do 35 ind3=1,fnumb
                   do 40 i=1,3
                      v1(i,1)=fcoord(i,ind3)
                      if (ind3.lt.fnumb) then
                         v2(i,1)=fcoord(i,ind3+1)
                      else
                         v2(i,1)=fcoord(i,1)
```

```
                        endif
40                   continue
c       Find the shortest vector connecting two edges.

                     call etest(v1,v2,type,vert1,vector,dist)

                     if (dist.lt.dmin) then
                        dmin=dist
                        do 45 i=1,3
                           pnorm(i)=vector(i)
                           pvert(i)=vert1(i)
45                      continue
                     endif
35                continue


          return
          end
```

```
        subroutine edgvrt(v1,v2,result,vert1,vector,dmin)

c       Given two edges which do not contain both endpoints of their common
c       normal, find the shortest normal distance from a vertex to an edge if
c       one exists. Otherwise, the shortest distance is a vertex-vertex
c       distance.
c          The shortest separating vector lies in the plane containing an
c       endpoint of the first (second) edge and normal to the second (first)
c       edge.  If no such vector is found then the shortest connecting vector
c       must join two endpoints and is normal to neither edge.

c       Given a point p1 and an edge with endpoints v1,v2, the projection
c       of p1 onto the edge is given by

c               p'=v1 + lambda*(v2-v1)

c       where:  lambda=(p1-v1)*(v2-v1)/(v2-v1)*(v2-v1)

c       The projection p' lies on the edge if 0.le.lambda.le.1.
c
c       Called from: etest

c       SUBROUTINES CALLED:
c               dotprd

c       PARAMETERS:
c               v1(3,2)-  the starting points of the two edges
c               v2(3,2)-  the end points of the two edges
c               result-   2- edge-vertex case
c                         3- vertex-edge case
c                         4- vertex-vertex case
c               vert1-    the closest point on the first edge
c               vector-   the shortest vector joining vert1 to the second edge
c               dist-     the shortest distance between the two edges.

c       Variables:
c               uedge-  an edge vector (v2-v1)
c

c*************************************************************************

        real v1(3,2),v2(3,2),vert1(3),vector(3),dist
        real p1(3),q1(3),uedge(3),udist(3),temp1(3),temp2(3),temp3(3)
        real lambda
        integer result
        logical stop

        dist=100000.
        result=4


c       Find the shortest connecting vector from each endpoint of one edge to
c       the second edge.

        do 25 ind1=1,2
           itemp=2
           if (indl.eq.2) itemp=1

c          Generate the line equation for an edge.

           do 5 i=1,3
              uedge(i)=v2(i,itemp)-vl(i,itemp)
              pl(i)=vl(i,itemp)
5          continue

c          Generate the other edge endpoints.

           do 20 ind2=1,2
              pdist=0.
              if (ind2.eq.1) then
                 do 10 i=1,3
                    ql(i)=vl(i,indl)
                    templ(i)=ql(i)-pl(i)
10               continue
              else
                 do 11 i=1,3
                    ql(i)=v2(i,indl)
                    templ(i)=ql(i)-pl(i)
11               continue
              endif

c             Calculate the projection of the point onto the edge.

              call dotprd(templ,uedge,temp2)
              call dotprd(uedge,uedge,temp3)
              lambda=temp2/temp3

c             If the intersection lies within the endpoints of the second edge,
c             then the shortest connecting vector joins the endpoint of the
c             first edge with the intersection point on the second edge.
c             If no intersections lie on an edge, the shortest connecting
c             vector joins two endpoints.

                 if (lambda.ge.0.and.lambda.le.1.) then

c                   Calculate the distance.

                    dist=0.
                    do 15 i=1,3
                       udist(i)=pl(i)+lambda*uedge(i)
                       dist=dist+(udist(i)-ql(i))**2
15                  continue
                    dist=sqrt(dist)

                    if (dist.lt.dmin) then
                       dmin=dist
                       result=3
                       if (indl.eq.2) result=2

c                      Edge-vertex case.

                       if (result.eq.2) then
```

```
                          do 17 i=1,3
                             vertl(i)=udist(i)
                             vector(i)=ql(i)-vertl(i)
17                        continue
                       else

c                      Vertex-edge case.

                          do 16 i=1,3
                             vertl(i)=ql(i)
                             vector(i)=udist(i)-ql(i)
16                        continue
                       endif
                    endif
                 endif

20         continue
25      continue

        return
        end
        subroutine editpa(lunit,idweld,index,values,changm)

c       Simulate and modify an inter-weld path.
c       The user can incrementally step in either direction along
c       the joint-interpolated path between two adjacent positions.
c       The robot and positioner can also be moved independently, joint
c       by joint. Positions can be added, removed or relocated.

c       Parameters:     lunit-  logical unit number for file operations
c                       idweld- the id of the weld associated with the path.
c                       index-  the first position of the path in the position
c                               table
c                       values- the joint values for a robot and table position

c       Subroutines called: addpos,movpos,rempos

c       Graphics calls: yes

c       Called from: dfpath

c       Variables:

c               delta-  the joint variable increments during motion
c               forwrd,revers-  flags for forward, reverse motion
c               jmove-  a logical array to selectively enable joint motion
c               nolim-  .true. if no limits exceeded
c               rmin,rmax-  physical joint limits
c               step-   .true. for single step motion
c               values- the robot and positioner joint values
c               changm- .true. if a joint change is made

c***************************************************************************

        real rmin(10,3),rmax(10,3)
        logical edit,add,rem,mov
        logical jmove(10,2)
        logical forwrd,revers
        logical start2
        real values(10,3)
        real delta(10,3)
        integer nrdof(3)
        logical step
        logical nolim
        logical changm

        common /jlimit/rmin,rmax
        common /nchain/nchain
        common /nrdof/nrdof
        common /jmove/jmove
        common /edit/edit,add,rem,mov
        common /direct/forwrd,revers
        common /start2/start2
        common /step/step


c       Set increment size for incremental motion of joints.
```

```
      do 1 i=1,nchain
         do 2 j=1,nrdof(i)
            delta(j,i)=5.
2        continue
1     continue

c     Move selected robot and table joints incrementally.
c     Motion can be forward or reverse, continuous or single step.
c     Motion cannot exceed joint limits.

      do 15 ind2=1,nchain
         do 10 ind1=1,nrdof(ind2)
            if (jmove(ind1,ind2)) then
               if (step) then
                  jmove(ind1,ind2)=.false.
               endif
               if (revers) then
                  temp=values(ind1,ind2)-delta(ind1,ind2)
               else
                  temp=values(ind1,ind2)+delta(ind1,ind2)
               endif
               if (temp.gt.180.) temp=temp-360.
               if (temp.lt.-180.) temp=temp+360.

               nolim=.false.
               if (rmin(ind1,ind2).le.rmax(ind1,ind2)) then
                  if (temp.le.rmax(ind1,ind2).and.temp.ge.rmin(ind1,ind2))
     *                then
                     nolim=.true.
                  endif
               else
                  if (temp.le.rmax(ind1,ind2).or.temp.ge.rmin(ind1,ind2)) the
                     nolim=.true.
                  endif
               endif

               if (nolim) then
                  changm=.true.
                  values(ind1,ind2)=temp
               else
                  call ringbe
                  call drtext('JOINT LIMIT EXCEEDED',' ')
               endif
            endif
10       continue
15    continue


c     Add a new robot or table position.

      if (add) then
         edit=.false.
         add=.false.
         start2=.true.
```

```
            call addpos(lunit,idweld,index,values)

c        Remove a robot or table position.

         elseif (rem) then
            edit=.false.
            rem=.false.
            start2=.true.
            call rempos(lunit,idweld,index)

c        Relocate a position.

         elseif (mov) then
            edit=.false.
            mov=.false.
            start2=.true.
            call movpos(lunit,index,values)

         endif

         return
         end
```

```
      subroutine etest(v1,v2,result,vert1,vector,dist)

c     Perform an edge-edge test to determine the minimum distance
c     between two edges. There are four possible cases:

c             1. the shortest connecting vector is the common normal, and
c             2. The shortest connecting vector starts at an endpoint
c                of the first edge and is normal to the second edge.
c             3. The shortest connecting vector starts at an endpoint
c                of the second edge and is normal to the first edge.
c             4. The shortest connecting vector connects endpoints of the
c                edges.

c     First, the common normal joining the lines containing the edges is
c     found.  The two intersection points are found.  Both, one or none of
c     these points lie on the two edges, giving the four cases above.


c     SUBROUTINES CALLED:
c             dotprd
c             edgvrt
c             solve
c             xprod
c             unit

c     Called from:  edges

c     PARAMETERS:
c             v1(3,2)-  the first vertices of the two edges
c             v2(3,2)-  the second vertices of the two edges
c             result-   1- edge-edge case
c                       2- edge-vertex case
c                       3- vertex-edge case
c                       4- vertex-vertex case
c             vert1-    the closest point on the first edge
c             vector-   the shortest vector joining the first and second edge
c             dist-     the shortest distance between the two edges
c
c***************************************************************************

      real v1(3,2),v2(3,2),vector(3),dist,vert1(3)
      real u(3,2)
      real p1(3),p2(3),u1(3),u2(3),temp(3),udiff(3)
      real lamb1,lamb2,diff(3),diff2(3,4),dtemp(4)
      integer result

      toler=.00001


c     define direction vectors for both edges

      do 10 i=1,3
         u(i,1)=v2(i,1)-v1(i,1)
         u(i,2)=v2(i,2)-v1(i,2)
         u1(i)=u(i,1)
         u2(i)=u(i,2)
```

```
           diff(i)=v1(i,2)-v1(i,1)
           p1(i)=v1(i,1)
           p2(i)=v1(i,2)
10      continue


c       find the common normal vector connecting edge 1 to edge 2, and find
c       the intersection points.

        call xprod(u1,u2,temp)

c       Test for singular condition where both lines are parallel.
c       If so, skip to the edge-vertex tests.

        tlen=0.
        do 11 i=1,3
           tlen=tlen+temp(i)**2
11      continue

        if (tlen.lt.toler) then
        write(*,*) 'tlen ',tlen
           lamb1=1000.
           lamb2=1000.
        else
           tlen=sqrt(tlen)
           do 12 i=1,3
              udiff(i)=temp(i)/tlen
12         continue
           call dotprd(diff,udiff,rlen)

           call solve(p1,p2,u1,u2,udiff,rlen,lamb1,lamb2)
        endif


        if (lamb1.ge.0..and.lamb1.le.1..and.lamb2.ge.0..and.lamb2.le.1.) then

c       Edge-edge case- the shortest vector connecting two lines intersects
c       both edges.


           result=1

           do 17 i=1,3
              vert1(i)=v1(i,1)+lamb1*u(i,1)
              vector(i)=diff(i)+lamb2*u(i,2)-lamb1*u(i,1)
17         continue

           dist=0.
           do 20 i=1,3
              dist=dist+vector(i)**2
20         continue
           dist=sqrt(dist)

        else

c       Edge-vertex (and vertex-edge) case- the shortest connecting vector
```

```
c       contains the endpoint of one edge and is normal to the other edge.

           call edgvrt(v1,v2,result,vert1,vector,dist)

        endif

           if (result.eq.4) then


c       Vertex-vertex case- the shortest connecting vector joins two edge
c       endpoints.


              do 60 i=1,4
                 dtemp(i)=0.
60            continue

c             Find all the vectors joining endpoints.

              do 65 i=1,3
                 diff2(i,1)=v1(i,2)-v1(i,1)
                 diff2(i,2)=v1(i,2)-v2(i,1)
                 diff2(i,3)=v2(i,2)-v1(i,1)
                 diff2(i,4)=v2(i,2)-v2(i,1)
                 dtemp(1)=dtemp(1)+diff2(i,1)**2
                 dtemp(2)=dtemp(2)+diff2(i,2)**2
                 dtemp(3)=dtemp(3)+diff2(i,3)**2
                 dtemp(4)=dtemp(4)+diff2(i,4)**2
65            continue

c             Search for the shortest joining vector.

              dist=100000000.
              do 70 i=1,4
                 if (dtemp(i).lt.dist) then
                    dist=dtemp(i)
                    isave=i
                 endif
70            continue

              do 75 i=1,3
                 vector(i)=diff2(i,isave)
75            continue

              if (isave.eq.1.or.isave.eq.3) then
                 do 80 i=1,3
                    vert1(i)=v1(i,1)
80               continue
              else
                 do 85 i=1,3
                    vert1(i)=v2(i,1)
85               continue
              endif
```

```
                dist=sqrt(dist)

             endif

          return
          end
```

```
        subroutine evpick(nsave,buffer,ntags,tags,istat)
c       Evaluate the objects hit by a pick.
c       Parameters:     nsave-  the number of picked objects to be saved.
c                       buffer- an array of picked objects returned from pick.
c                       ntags-number of tags
c                       tags- tags(1)- tag of objects
c                             tags(2,3)- first and last offset of objects
c                       istat- status (future use)
c       Subroutines called: none
c       Called from: mkseam
c       Variables:
c               index-  pointer to the current pick entry in 'buffer'
c               last-   the previous object tag
c               max-    the largest offset from a tag
c               min-    the smallest offset from a tag
c               nname-  the number of tags counted
c               next-   the current object tag
c
c*****************************************************************************

        integer buffer(*)
        integer ntags,tags(3,*)

        index=1
        ntags=0
        istat=0
        last=-1
        nname=1

100     continue
        if (buffer(index+2).eq.-1.or.nname.gt.nsave) goto 200
        if (buffer(index).ne.-1) then
           next=buffer(index+1)
           if (next.ne.last) then
              ntags=ntags+1
              min=buffer(index+2)
              max=min
              tags(1,ntags)=next
              tags(2,ntags)=min
              tags(3,ntags)=max
              last=next
           else
              max=buffer(index+2)
              tags(3,ntags)=max
           endif
        endif
        index=index+3
        nname=nname+1
        goto 100
200     continue

        return
        end
        subroutine exprt1(wsize,jtype,xsect,t1,t2,speed,alpha,beta,stkout)
c       Calculate the best geometric welding parameters as a function of
c       given parameters.

c       Called from: wgeom

c       Subroutines called: none

c       Parameters:
c                 wsize-  wire diameter (inch)
c                 jtype-  joint type
c                 xsect-  weld cross sectional area (sq. in.)
c                 t1-     thickness of first part (inch)
c                 t2-     thickness of second part (inch)
c                 speed-  welding speed (inch/min)
c                 alpha-  torch longitudinal angle (deg.)
c                 beta-   torch transverse angle (deg.)
c                 stkout- torch gap (inch)

c       Note: these calculations are based on imperial units.  No conversion
c       factors have been incorporated.

c*******************************************************************************

        real xsect,t1,t2,speed,alpha,beta,stkout
        integer jtype


        tx=amin1(t1,t2)

c       Wire size...  .030 inch.

        if (wsize.eq..030) then

           feed=290.38+52.87*alog(tx)
           speed=3.1415*(.030**2)*feed/xsect
           stkout=.75+.25*alog(tx)
           alpha=15.
           beta=0.


c       Wire size...  .035 inch.

        elseif (wsize.eq..035) then

           feed=435.77+101.51*alog(tx)
           if (feed.gt.265.) feed=265.
           speed=3.1415*(.035**2)*feed/(xsect*4.)
           stkout=.75+.25*alog(tx)
           alpha=15.
           beta=0.


c       Wire size...  .045 inch.
```

```
        elseif (wsize.eq..045) then

           feed=296.88+21.77*alog(tx)
           speed=3.1415*(.045**2)*feed/(xsect*4.)
           stkout=.75+.25*alog(tx)
           alpha=15.
           beta=0.

c       else error

        endif

c       Convert speed from inch/min  to mm/sec.

        speed=speed*(.4233)

        return
        end
```

```
      subroutine exprt2(jtype,xsect,t1,t2,T,A)

c     Calculate the best geometric welding parameters as a function of
c     given parameters.

c     Called from: wparam

c     Subroutines called: none

c     Parameters:
c              jtype-  joint type
c              xsect-  weld cross sectional area (sq.in.)
c              t1-     thickness of first part (inch)
c              t2-     thickness of second part (inch)
c              T-      seam tangent vector in world coords.
c              A-      seam normal vector in world coords.

c     Variables:
c              volt-   welding voltage (volts)
c              feed-   wire feed rate (inch/min)
c              gfr-    gas flow rate (cfm)
c              currnt- welding current (amps)

c     Note: these calculations are based on imperial units, and
c           no conversion factors have been incorporated.

c*****************************************************************************

      real xsect,t1,t2,speed,alpha,beta,stkout
      real T(3),A(3)
      integer jtype

      common /exprt/volt,wsize,feed,gfr,walpha,beta,stkout
      common /currnt/currnt

      tx=amin1(t1,t2)

c     Wire size...  .030 inch.

      if (wsize.eq..030) then

         volt=16.
         feed=290.38+52.87*alog(tx)
         gfr=23.38+1.64*alog(tx)
         currnt=199.07+43.04*alog(tx)


c     Wire size...  .035 inch.

      elseif (wsize.eq..035) then

         feed=435.77+101.51*alog(tx)
         if (feed.gt.265.) feed=265.
         volt=23.91+2.12*alog(tx)
         gfr=40.55+7.17*alog(tx)
         if (gfr.gt.22.5) gfr=22.5
```

```
            currnt=222.64+44.41*alog(tx)
            if (currnt.gt.150.) currnt=150.


c       Wire size...  .045 inch.

        elseif (wsize.eq..045) then

            feed=296.88+21.77*alog(tx)
            volt=22.
            gfr=22.5
            currnt=207.88+8.72*alog(tx)

c       else error

        endif

c       Adjust the current as a function of the weld orientation relative
c       to the gravity vector.

        call calcs(T,A,cphi,cthe)

        currnt=currnt*cphi*cthe


        return
        end
```

```
      subroutine finbes(pstat,face,vclos,pvert,pnorm,dmin)

c     Find the best separating plane between two parts. This plane is
c     tangent to one part and its distance from the other part is
c     the minimum separating distance.

c     Called from: plist

c     SUBROUTINES CALLED:
c             ajvrts
c             cdist
c             edges
c             ftest
c             getcrd
c             getnml
c             getvrt
c             pface
c             vrtlst

c     PARAMETERS:
c             pstat-  .true. if the two parts have parallel opposing faces
c             face-   the ID of the best separating face
c             vclos-  the ID of the closest vertex if pstat=.false.
c                     else the ID of the parallel opposing face
c             pvert-  a vector lying on the separating plane
c             pnorm-  the surface normal of the separating plane
c             dmin-   the minimum distance between the two parts

c***********************************************************************

      real fcoord(3,20)
      integer vlist(20),vprev(50),vnext(50)

      integer face,cvert,vclos,vnumb,fnumb
      real pnorm(3),dist,temp1(3),temp2(3)
      real pvert(3),coord1(3),coord2(3),vector(3)
      real normal(3),norvec(3),vface(3)
      integer verts(2),faces(2),istart(2),iend(2)
      integer nprev
      logical result,stop,pstat,flag,flag2

      toler=1.
      toler2=.0001

      cvert=vclos


c     If two opposing faces are parallel, test if any vertex of one face
c     projects onto the other face within its boundaries.

      if (pstat) then
         faces(1)=face
         faces(2)=vclos
         call vrtlst(faces(1),istart(1),iend(1))
```

```
           call vrtlst(faces(2),istart(2),iend(2))
           result=.false.

c          Test each vertex of both faces against the other face until a
c          vertex is found which projects onto a face.

           do 50 indl=1,2
              if (.not.result) then
                 if (indl.eq.1) ind2=2
                 if (indl.eq.2) ind2=1
                 do 55 ind3=istart(ind2),iend(ind2)
                    if (.not.result) then
                       call getvrt(ind3,ivert)
                       call pface(faces(indl),ivert,vface,dmin)
                       call ftest(faces(indl),vface,result)

c                      Calculate the resulting separating vector.

                       if (result) then
                          call getcrd(ivert,pvert)
                          call getnml(face,pnorm)
                          do 11 i=1,3
                             pnorm(i)=pnorm(i)*dmin
                             if (indl.eq.1) then
                                pvert(i)=pvert(i)-pnorm(i)
                             endif
11                        continue
                       endif

                    endif
55               continue
              endif
50         continue

           if (.not.result) then
              call getvrt(istart(2),cvert)
           endif
        else

c       Test if the closest vertex projects onto the test face.

           call pface(face,cvert,vface,dmin)
           call ftest(face,vface,result)

c          If it does, calculate the separating vector

           if (result) then
              call getcrd(cvert,pvert)
              call getnml(face,pnorm)
              do 13 i=1,3
                 pnorm(i)=pnorm(i)*dmin
                 pvert(i)=pvert(i)-pnorm(i)
13            continue
           endif
        endif


c       If a closest vertex projects onto a face, then the minimum separating
c       distance is the vertex-face distance. Otherwise, edge-edge tests
c       must be done to determine the shortest distance.

        if (.not.result) then

c          Generate a list of the vertices on the face.

           call vrtlst(face,jstart,jend)
           call getnml(face,normal)
           fnumb=0
           do 10 indl=jstart,jend
              fnumb=fnumb+1
              call getvrt(indl,ipoint)
              call getcrd(ipoint,coordl)
              do 15 i=1,3
                 fcoord(i,fnumb)=coordl(i)
15            continue
10         continue


           dmin=10000.
           stop=.false.
           nprev=0
           nvnext=0

c       Generate a set of edges by connecting the current vertex to each
c       topologically adjacent vertex.  Perform an edge-edge test of each
c       of these edges against all of the edges of the test face.
c       Repeat until all the edges have been tested against the test face.
c       Save the minimum separating distance, and the separating vector.

c       WHILE (not the minimum distance)

100        if (stop) goto 200

c             Add the current starting vertex to the list of previous
c             starting vertices.

              stop=.false.
              nprev=nprev+1
              vprev(nprev)=cvert
              call getcrd(cvert,coordl)

c             Test all the edges connecting the current vertex to its adjacent
c             vertices against the edges of the face.

              call ajvrts(cvert,vlist,vnumb)
              do 80 indll=1,vnumb
                 flag=.true.
                 flag2=.true.

c                Edges adjacent to previously tested vertices have already been
```

```
c                tested.

                 do 85 ind22=1,nprev
                    if (vlist(indll).eq.vprev(ind22)) then
                       flag=.false.
                    endif
85               continue

c                Test each new edge against all the face edges.

                 if (flag) then
                    call getcrd(vlist(indll),coord2)

c                   Perform edge-edge tests. 'pnorm','pvert','dmin' are
c                   updated if dmin is a new minimum.

                    call edges(coordl,coord2,fcoord,fnumb,pnorm,pvert,dmin)

c                   Put all the edge endpoints into a queue, to be tested as
c                   starting points.

                    do 90 ind33=1,nvnext
                       if (vlist(indll).eq.vnext(ind33)) then
                          flag2=.false.
                       endif
90                  continue
                    if (flag2) then
                       nvnext=nvnext+1
                       vnext(nvnext)=vlist(indll)
                    endif
                 endif
80            continue

c             If there are any entries left, pick the new starting point from
c             the queue.

              if (nvnext.gt.0) then
                 cvert=vnext(nvnext)
                 nvnext=nvnext-1
              else
                 stop=.true.
              endif

           goto 100
200        continue


c       Test if a separating plane was found.

c       Find the plane equation parameters for the separating plane.

        scale=0.
        fdist=0.
        do 42 i=1,3
           scale=scale+pnorm(i)**2
           fdist=fdist+pvert(i)*pnorm(i)
```

```
42      continue
        scale=sqrt(scale)
        if (scale.gt.toler2) then
           do 43 i=1,3
              norvec(i)=pnorm(i)/scale
43         continue
           fdist=fdist/scale
        endif

c       Test if all vertices of the first part lie 'inside' the plane and
c       all vertices of the second part lie 'outside' the plane.
c       If so, the plane is a separating plane and the parts are disjoint.

        call getvrt(jstart,ivert)
        call cdist(2,norvec,fdist,ivert,tdist)
        if (tdist.lt.toler) then
           call cdist(1,norvec,fdist,cvert,dtemp)
           if (dtemp.lt.0. ) dmin=-1.
        else
           dmin=-1.
        endif

        endif

        return
        end
        subroutine finsep(type,objcts,index,bestf,bestv,status,pstat)

c       Find the best separating plane between two convex polyhedral objects.
c       This algorithm searches for a face which defines a separating plane.
c       If none exist, the face which comes closest is returned, along with
c       the nearest vertex of the other object.


c       Called from: plist

c       SUBROUTINES CALLED:
c               adjcnt
c               facvrt
c               getbes
c               getcrd
c               getnml
c               putbes

c       PARAMETERS:
c               type-   0- find any separating plane
c                       1- find the best separating plane
c               objcts- the two polyhedral objects to be compared
c               index-  indicates which object the returned face belongs to
c               bestf-  ID of the returned face
c               bestv-  ID of the corresponding closest vertex of the other
c                       object. If two faces are parallel (pstat=.true.) then
c                       'bestv' is the other face ID.
c               status- 0- a separating face was found
c                       1- no separating face was found
c               pstat-  .true. if two opposing faces are parallel

c       Variables:
c               cface-  the current test faces
c               cvert-  the current test vertices
c               flist-  list of faces adjacent to a face
c               fprev-  list of previous adjacent faces
c               fstart- the starting face
c               gtdmin- the largest minimum vertex to face distance for two
c                       parts
c               ntbest- .true. if the current face is not the best separating
c                       face
c               ntclos- .true. if the current vertex is not the closest vertex
c                       to the current face
c               objcts- the pair of parts being tested
c               startf- .true. for the starting face
c               startv- .true. for the starting vertex
c               vlist-  list of vertices adjacent to a vertex
c               vprev-  list of previous adjacent vertices
c               vstart- the starting vertex
c*****************************************************************************

        integer flist(20),fprev(50),vlist(20),vprev(50)

        integer type,objcts(2)
        real dist,normal(3),coord(3)
```

```
      real dmin,gtdmin(2),fdist,vdist
      integer cface(2),cvert(2),cvsave(2)
      integer bestf,bestv,status
      integer fnumb,nfprev,vnumb,nvprev
      integer fstart,vstart
      logical stop,ntbest,ntclos,startf,startv,pstat


c     Initialize parameters.

      stop=.false.
      pstat=.false.
      status=0
      toler=.00001
      fnumb=0
      nfprev=0
      vnumb=0
      nvprev=0
      gtdmin(1)=0.
      gtdmin(2)=0.
      cface(2)=0
      cvert(2)=0

c     use the previous best face and vertex as the current starting face
c     and vertex

      call getbes(1,objcts,fstart,vstart)


c     initialize the current face and vertex

      cface(1)=fstart
      cvert(1)=vstart


c     test each object against the other

      do 10 ind1=1,2
         if (.not.stop) then
         ntbest=.true.
         fnumb=0
         nfprev=0
         gtdmin(ind1)=-1000.
         startf=.true.


c     scan all of the faces until the best one is found

c        WHILE (ntbest)

100      if (.not.ntbest) goto 200


c     Find the set of faces adjacent to the current face.
c     The best face is the one with the greatest minimum distance
c     to a vertex of the other object. If all the adjacent faces have
```

```
c       already been tested, then the current face is the best face.

              call adjcnt(1,cface(ind1),flist,fnumb,fprev,nfprev)
              if (fnumb.eq.1) ntbest=.false.
              if (ntbest) then

c             At the start, process the current face.  After, test only the
c             adjacent faces.

              if (startf) then
                 init=1
              else
                 init=2
              endif

              ntbest=.false.
              do 15 ind2=init,fnumb
                 if (.not.stop) then
                 call getnml(flist(ind2),normal)
                 call facvrt(flist(ind2),coord)
                 fdist=0.
                 do 16 i=1,3
                    fdist=fdist+coord(i)*normal(i)
16               continue

                 ntclos=.true.


c       Find the vertex of the other object which is closest to the current face
c       plane.

                 nvprev=0
                 vnumb=0
                 dmin=10000.
                 startv=.true.

c                WHILE (ntclos)

101              if (.not.ntclos) goto 201


c       Find the vertices adjacent to the current vertex
c       and test which one is closest to the face plane.  That vertex
c       becomes the new current vertex.

                    call adjcnt(2,cvert(ind1),vlist,vnumb,vprev,nvprev)
                    if (vnumb.eq.1) ntclos=.false.
                    if (ntclos) then

c                      Test the current vertex if it is the first one.

                       itest=0
                       if (startv) then
                          init=1
                       else
                          init=2
```

```
                      endif

c                     Calculate the distance of each vertex from the current
c                     face, and update the current closest vertex.

                      do 20 ind3=init,vnumb
                         call getcrd(vlist(ind3),coord)
                         vdist=0.
                         do 21 i=1,3
                            vdist=vdist+coord(i)*normal(i)
21                       continue
                         vdist=vdist-fdist

                         if (vdist.lt.dmin) then
                            dmin=vdist
                            cvert(ind1)=vlist(ind3)
                            ntclos=.true.
                            itest=1
                         endif

20                    continue
                      if (itest.ne.1) ntclos=.false.

                   endif

                   startv=.false.

c               END WHILE
                go to 101
201             continue

c       stop if a separating face has been found

                if (dmin.ge.0..and.type.eq.0) then
                   ntbest=.false.
                   stop=.true.
                endif

c       update the best separating face

                if (fnumb.eq.1) ntbest=.false.
                if (ind2.eq.1) then
                   gtdmin(ind1)=dmin
                   cvsave(ind1)=cvert(ind1)
                else
                   if (dmin.gt.gtdmin(ind1)) then
                      gtdmin(ind1)=dmin
                      cvsave(ind1)=cvert(ind1)
                      cface(ind1)=flist(ind2)
                      ntbest=.true.
                   endif
                endif
             endif
15           continue

             endif
```

```
              startf=.false.
c          END WHILE
           go to 100
200        continue
        endif

c       When the objects are reversed, choose the new current face to
c       be adjacent to the current vertex, and the new current vertex to
c       belong to the current face.  These starting points should be
c       close to the optimum, reducing the search required.

        if ((indl.eq.1).and.(.not.stop)) then
           call getbes(2,objcts,cface(2),cvert(2))
        endif

10      continue

c       determine which object has the best separating face
c       or if there are two parallel faces

        diff=gtdmin(1)-gtdmin(2)
        if (diff.gt.toler) then
           index=1
           bestf=cface(1)
           bestv=cvsave(1)
           call putbes(1,objcts,bestf,bestv)
        elseif (diff.lt.-toler) then
           index=2
           bestf=cface(2)
           bestv=cvsave(2)
           call putbes(2,objcts,bestf,bestv)
        else
           index=1
           bestf=cface(1)
           bestv=cface(2)
           pstat=.true.
        endif


c       If the greatest minimum distance from a vertex to a face plane is
c       negative, then none of the faces define a separating plane, and
c       further testing is needed.

        if (gtdmin(index).lt.0) status=1

        return
        end
```

```
c       Frame to frame coherence module.


c       Subroutines:    getbes,initfm,putbes


c       This module employs frame to frame coherence of the scene to reduce
c       search. The closest face and vertex of each pair of objects are stored,
c       and are used as a starting point for the search during the next frame
c       update.


c*************************************************************************

        subroutine initfm(number)

c       Initialize the stored face and vertex list by arbitrarily inserting
c       the first face and vertex of each part.

c       Parameters:
c               number- the number of parts

c       Called from:

c       Subroutines called: fplist,getvrt,vrtlst

c       Variables:
c               lstbes- the previous best separating face and closest vertex
c                       of each pair of parts
c               nparts- the number of parts
c               parts-  list of parts

c*************************************************************************

        integer*2 lstbes(2,10000),nparts
        integer parts(100)

        common /last/lstbes,nparts

        max=100
        nparts=number

        do 30 ind=1,nparts
           if (ind.le.max) then
              parts(ind)=ind
           else
              write(*,*) 'Maximum number of parts exceeded in INITFM'
           endif
30      continue

        index=0
        do 10 ind1=1,nparts
           call fplist(parts(ind1),iface,iend)
           do 20 ind2=1,nparts
```

```
              index=nparts*(parts(ind1)-1)+parts(ind2)
              if (ind1.ne.ind2) then
                 call fplist(parts(ind2),istart,iend)
                 call vrtlst(istart,itemp,iend)
                 call getvrt(itemp,ivert)
                 lstbes(1,index)=iface
                 lstbes(2,index)=ivert
              endif
20         continue
10      continue

        return
        end

c*******************************************************************************

        subroutine putbes(ind,objcts,fstart,vstart)

c       Store the new best separating face and closest vertex for a pair of
c       parts.

c       Parameters:
c               ind- 1- best face belongs to first part
c                    2- best face belongs to second part
c               objcts- a pair of parts
c               fstart- best separating face id
c               vstart- closest vertex of other part

c       Called from: finsep

c       Variables:
c               lstbes- the previous best separating face and closest vertex
c                       of each pair of parts

c*******************************************************************************

        integer objcts(2),fstart,vstart
        integer*2 lstbes(2,10000),nparts

        common /last/lstbes,nparts

        if(ind.eq.1) then
           index=nparts*(objcts(1)-1)+objcts(2)
        else
           index=nparts*(objcts(2)-1)+objcts(1)
        endif

        lstbes(1,index)=fstart
        lstbes(2,index)=vstart

        return
        end

c*******************************************************************************
```

```
      subroutine getbes(ind,objcts,fstart,vstart)
c     Get the previous best separating plane and closest vertex for a pair
c     of parts.

c
c     Parameters:
c             ind- 1- best face belongs to first part
c                  2- best face belongs to second part
c             objcts- a pair of parts
c             fstart- best separating face id
c             vstart- closest vertex of other part

c     Called from: finsep

c     Variables:
c             lstbes- the previous best separating face and closest vertex
c                     of each pair of parts
c*****************************************************************************

      integer objcts(2),fstart,vstart
      integer*2 lstbes(2,10000),nparts

      common /last/lstbes,nparts

      if(ind.eq.1) then
         index=nparts*(objcts(1)-1)+objcts(2)
      else
         index=nparts*(objcts(2)-1)+objcts(1)
      endif

      fstart=lstbes(1,index)
      vstart=lstbes(2,index)

      return
      end
```

```
        subroutine ftest(face,vert,result)

c       This subroutine tests if a point on a face plane lies
c       within the boundaries of the face.

c       SUBROUTINES CALLED:
c               getcrd
c               getvrt
c               vrtlst

c       Called from: finbes


c       PARAMETERS:
c               face-   the ID of the test face.
c               vert-   the coordinates of the test point
c               result- .true. if the point lies within the face boundaries


c       Variables:

c               chaxes- .true. if a change of test axis is required.
c               delta-  a small perterbation constant
c               dmax-   the maximum distance of a vertex from the test plane
c               dist,lstdst-  the normal distances of vertices from the plane
c               flist-  the first vertices of face edges which intersect the
c                       test plane
c               iaxis-  the normal axis
c               lstcrd,ecoord- the previous and current vertex coordinates.
c               rcoord- the coordinates of the edge intersection points along
c                       the test axes
c               singul- .true. if a singularity is found
c               test-   the coordinate of the test point along the axis
c                       normal to the test plane
c               unorm-  an intersecting edge vector

c**************************************************************************

        integer face,flist(2)
        real dist,lstdst
        real vert(3),lstcrd(3),ecoord(3)
        real coord1(3),coord2(3),rcoord(2),unorm(3)
        logical singul,chaxes,result


c       initialize parameters

        toler=.00001
        delta=.0001
        rtoler=.00001
        iaxis=1
        singul=.true.
        chaxes=.true.
        result=.false.
        dmax=-10000.

        call vrtlst(face,istart,iend)


c       Define a plane normal to one of the principal axes and containing the
c       test point.
c       If the face and the test plane are not parallel and if exactly two
c       edges of the face intersect the plane, then the point may be contained
c       by the face.
c       If the number of intersections is zero, then
c       the point lies outside the face.  If the number of intersections is
c       not 0 or 2, then we have a singularity condition which can be avoided
c       by perturbing the location of the point slightly.

        do 10 ind1=1,3
           if (singul.or.chaxes) then
              chaxes=.false.
              test=vert(iaxis)
              icount=0
              call getvrt(istart,ipoint)
              call getcrd(ipoint,lstcrd)
              lstdst=lstcrd(iaxis)-test
              do 20 ind2=istart,iend
                 if (ind2.lt.iend) then
                    call getvrt(ind2+1,ipoint)
                    call getcrd(ipoint,ecoord)
                 else
                    call getvrt(istart,ipoint)
                    call getcrd(ipoint,ecoord)
                 endif
                 dist=ecoord(iaxis)-test

c                Test if previous and current vertices are on opposite sides of
c                the test plane. Update the maximum distance.

                 if (dist*lstdst.le.0.) then
                    icount=icount+1
                    if (icount.le.2) flist(icount)=ind2
                 endif
                 dtemp=abs(dist)
                 if (dtemp.gt.dmax) dmax=dtemp

                 lstdst=dist
                 do 25 i=1,3
                    lstcrd(i)=ecoord(i)
25               continue
20            continue

c             If the maximum distance is too small, the face lies nearly on the
c             test plane and a different test plane should be tried.

              if (dmax.lt.toler) then
                 chaxes=.true.
                 iaxis=2
```

```
c             If a singularity condition occurs, move the test point slightly
c             and try again.

              elseif (icount.eq.0.or.icount.eq.2) then
                 singul=.false.
              else
                 test=test+delta
              endif
           endif
10      continue

c       If the face intersects the plane, then we can find the coordinates of
c       the two intersection points. These points lie on the plane and are
c       collinear with the test point.  If we define a pair of axes aligned
c       with the two principal axes lying in the plane, and whose origin is
c       the test point, then the test point lies outside of the face if
c       the two face intersection points lie in the same quadrant of the
c       plane. We only need to test one component of the intersection points
c       to see if they cross one of the axes.

        iaxis2=iaxis

        if (icount.eq.2) then
           chaxes=.true.
           do 30 ind1=1,2
              if (chaxes) then
                 chaxes=.false.
                 if (iaxis2.le.2) then
                    iaxis2=iaxis2+1
                 else
                    iaxis2=1
                 endif

c                For each intersecting edge, calculate its intersection point
c                and test if their coordinates along the test axis have the
c                same sign.

                 do 35 ind2=1,2
                    call getvrt(flist(ind2),ivert)
                    if (flist(ind2).eq.iend) then
                       call getvrt(istart,inext)
                    else
                       call getvrt(flist(ind2)+1,inext)
                    endif
                    call getcrd(ivert,coord1)
                    call getcrd(inext,coord2)
                    do 36 i=1,3
                       unorm(i)=coord2(i)-coord1(i)
36                  continue

c                   Test for a face edge lying on the test plane.

                    if (abs(unorm(iaxis)).lt.toler) then
                       rcoord(ind2)=coord1(iaxis2)
                    else
                       rlamb=(vert(iaxis)-coord1(iaxis))/unorm(iaxis)
                       rcoord(ind2)=coord1(iaxis2)+rlamb*unorm(iaxis2)
                       rcoord(ind2)=rcoord(ind2)-vert(iaxis2)
                    endif
35               continue

                 rtest=rcoord(1)*rcoord(2)
                 if (rtest.lt.-rtoler) then
                    result=.true.
                 else
                    chaxes=.true.
                 endif

              endif
30         continue
        endif

        return
        end
```

```
      subroutine genpat(idseam,files,nfiles,ipick,icount)

c     Generate a welding trajectory for a given seam.

c     Subroutines called: config,drtext,welder

c     Graphics calls: ringbe

c     Called from: smpick

c     Parameters:

c             idseam- the id of the seam to be welded
c             files-  CAD segments defining the seam
c             nfiles- the number of segments
c             ipick-  the list of welded seams
c             icount- the number of welded seams

c***********************************************************************************

      integer idseam,files(3,*),nfiles,ipick(*),icount
      logical update
      logical start

      common /update/update
      common /start/start
      common /npoint/npoint


c     Calculate the number of points along the weld.

      npoint=0
      do 10 i=1,nfiles
         istart=files(2,i)
         iend=files(3,i)
         npoint=npoint+abs(iend-istart)+1
10    continue

c     Generate geometric welding data from CAD specifications.

      call wgeom(idseam,files,nfiles)

c     Search for a feasible welding path.

      call config(idseam,files,nfiles,istat)

      start=.true.

      if (istat.eq.0) then

c        If a feasible path was found, use the results to calculate
c        the remaining welding process parameters.

         call wparam(idseam,files,nfiles)

      else
```

```
c          Else remove the seam from the list of welds.

           ipick(icount)=0
           icount=icount-1
           update=.true.

c          Status messages.

           if (istat.eq.1) then
              call ringbe
              call drtext('ROBOT ENVELOPE EXCEEDED- REPOSITION  ',' ')
           elseif (istat.eq.2) then
              call ringbe
              call drtext('JOINT LIMITS EXCEEDED- REPOSITION  ',' ')
           elseif (istat.eq.3) then
              call ringbe
              call drtext('JOINT CHANGE TOO LARGE',' ')
           elseif (istat.eq.4) then
              call ringbe
              call drtext('EXIT WELD GENERATION',' ')
           endif

        endif

        return
        end
        subroutine getsep(ipart,icount,imax)

c       Return the starting location in the separating plane table
c       corresponding to the given part.

c       PARAMETERS:
c               ipart-  the part number
c               icount- the location of the first separating plane
c               imax-   the location of the last separating plane

c       Variables:

c               ptable- pointer table containing the starting index and number
c                       of entries in the separating plane list for a given
c                       part

c       Called from: plist,trandb

c********************************************************************************

$include seplane.com

        icount=ptabl(ipart)

        imax=ptabl(ipart+1)-1

        return
        end
        SUBROUTINE HMULT(A,B,C)
C
C       THIS SUBROUTINE MULTIPLIES TWO 4X4 HOMOGENEOUS TRANSFORMATIONS.
C               C=AB,  WHERE A, B, C ARE HOMOGENEOUS TRANSFORMATIONS.
C
c       The IRIS hardware matrix operations are used.
c
c       Called from: init,worpos

c       Graphics calls: yes
C
C**********************************************************************
C
        REAL A(4,4),B(4,4),C(4,4)
C
        call pushma
        call loadma(A)
        call multma(B)
        call getmat(C)
        call popmat

        RETURN
        END
```

```
      SUBROUTINE INIT
C
C     THIS SUBROUTINE INITIALIZES THE VALUES OF ALL CONSTANT
C     PARAMETERS WHICH ARE STORED IN FILES AND SAVES THEM IN
C     COMMON BLOCKS.
C
C     INPUTS: none
C
C     OUTPUTS: NONE
C
C
C     FILES READ: WKSTAT- ROBOT DESCRIPTION
C
C
c     Called from: main

C     SUBROUTINES CALLED:     MATINV, hmult
C
C***************************************************************************
C
      CHARACTER*20 WKSTAT
      REAL HTW(4,4),HPRE(4,4),HPOST(4,4)
      REAL ALPHA(10,3),LEN(10,3),THETA(10,3),DIST(10,3),RMIN(10,3),RMAX(10,3),
     * H00(4,4),H6G(4,4),
     * INH00(4,4),INH6A(4,4)
      INTEGER LINK(10,3),JTYPE(10,3),NRDOF(3)
      REAL DELTA
      INTEGER ICONF(6),IMAX
      REAL HPRE1(4,4),HPOST1(4,4),HPRE2(4,4),HPOST2(4,4)
      real minmov(10,3), mincha(10,3),maxjv(10,3)
      integer workpc
C
      COMMON /ROBOT/LINK,JTYPE,ALPHA,LEN,THETA,DIST
      common /jlimit/rmin,rmax
      common /nchain/nchain
      common /nrdof/nrdof
      COMMON /TRANS/HPRE1,HPOST1,HPRE2,HPOST2
      COMMON /INVERS/INH00,INH6A
      COMMON /WKSTAT/WKSTAT
      COMMON /TOLROT/IMAX,DELTA
      COMMON /CONFIG/ICONF
      common /minmov/minmov
      common /mincha/mincha
      common /maxjv/maxjv
      common /workpc/workpc
C
C
C*****READ FIXED ROBOT PARAMETERS   ****************************************
C
      open(unit=1,file=wkstat,status='old')

      READ(1,*)

C*****READ THE NUMBER OF KINEMATIC CHAINS

      READ(1,*) NCHAIN
```

```
C
        READ(1,*)
C
C*****  READ THE NUMBER OF DEGREES OF FREEDOM OF EACH CHAIN
C
        READ(1,*) (NRDOF(I),I=1,NCHAIN)
C
C*****  READ THE ROBOT GEOMETRY PARAMETERS AND JOINT LIMITS
C
        DO 5 J=1,NCHAIN
           READ(1,*)
           READ(1,*)
           DO 10 I=1,NRDOF(J)
              READ(1,*) LINK(I,J),JTYPE(I,J),ALPHA(I,J),LEN(I,J),THETA(I,J),
     *        DIST(I,J),RMIN(I,J),RMAX(I,J)
10         CONTINUE
5       CONTINUE

c*****  Read the cluster id of the workpiece.

        read(1,*)
        read(1,*) workpc
C
C*****  READ INITIAL ROBOT CONFIGURATION
C
        READ(1,*)
        READ(1,*) (ICONF(I),I=1,6)
C
C*****  READ THE WORLD TO ROBOT BASE HOMOGENEOUS TRANSFORMATION 'H00'
C
        READ(1,*)
        DO 20 I=1,4
C          READ(1,*) (H00(I,J),J=1,4)
           READ(1,*) (HPRE1(I,J),J=1,4)
20      CONTINUE
C
C*****  READ THE TOOL TO LAST LINK HOMOGENEOUS TRANSFORMATION 'H6G'
C
        READ(1,*)
        DO 30 I=1,4
C          READ(1,*) (H6G(I,J),J=1,4)
           READ(1,*) (HPOST1(I,J),J=1,4)
30      CONTINUE

c*****  READ THE TABLE TO WORKPIECE TRANSFORMATION 'HTW'

        READ(1,*)
        DO 100 I=1,4
           READ(1,*) (HTW(I,J),J=1,4)
100     CONTINUE

C*****  READ THE TABLE TRANSFORMATION 'HPRE'

        READ(1,*)
        DO 101 I=1,4
           READ(1,*) (HPRE2(I,J),J=1,4)
```

```
101     CONTINUE

C*****  READ THE TABLE TRANSFORMATION 'HPOST'.

        READ(1,*)
        DO 102 I=1,4
           READ(1,*) (HPOST(I,J),J=1,4)
102     CONTINUE

c*****  Calculate the combined post transformation.

        call hmult(hpost,htw,hpost2)
C
C*****  READ THE NUMBER AND SIZE OF THE DESIRED ROTATIONS ABOUT THE TOOL
C*****  AXIS (6 DEG. OF FREEDOM ROBOT ONLY)
C
        READ(1,*)
        READ(1,*) IMAX,DELTA
        DELTA=DELTA*3.14159/180.

c       Read the maximum joint value changes.

        read(1,*)
        do 120 i=1,nchain
           read(1,*) (minmov(j,i),j=1,nrdof(i))
120     continue

c       Read the minimum joint value changes for joint interpolated motion.

        read(1,*)
        do 125 i=1,nchain
           read(1,*) (mincha(j,i),j=1,nrdof(i))
125     continue
C
c       Read the maximum joint velocities.

        read(1,*)
        do 130 i=1,nchain
           read(1,*) (maxjv(j,i),j=1,nrdof(i))
130     continue

        close(1)
C
C
C*****  CALCULATE INVERSE MATRICES FOR USE LATER.
C
        call matinv(hprel,inho0)
        call matinv(hpost1,inh6a)
C
C
        RETURN
        END
        subroutine initgr

c       Initialize the graphics environment.

c       Subroutines called: none

c       Graphics calls: yes

c       Called from: main

c*******************************************************************************
c
C
$INCLUDE /usr/include/fdevice.h
$INCLUDE /usr/include/fgl.h

C
        integer*2 icolor,imap

c       initialize graphics
c
        CALL GINIT
C

c       Set up double buffer mode and map a grey scale and some colours.

        CALL DOUBLE
        call gconfi
        do 80 imap=0,11
           icolor=21*imap
        CALL MAPCOL(imap,icolor,icolor,icolor)
80      continue
        call mapcol(i2(12),i2(255),i2(0),i2(0))
        call mapcol(i2(13),i2(0),i2(255),i2(0))
        call mapcol(i2(14),i2(255),i2(0),i2(255))
        call mapcol(i2(15),i2(255),i2(255),i2(0))


        call swapbu
        do 81 imap=0,11
           icolor=21*imap
           call mapcol(imap,icolor,icolor,icolor)
81      continue
        call mapcol(i2(12),i2(255),i2(0),i2(0))
        call mapcol(i2(13),i2(0),i2(255),i2(0))
        call mapcol(i2(14),i2(255),i2(0),i2(255))
        call mapcol(i2(15),i2(255),i2(255),i2(0))

c       define cursor color as yellow

        call setcur(i2(0),i2(15),i2(15))

c       set up buttons and valuators

        call QBUTTO(mousel)
        call qbutto(padpfl)
```

```
      call qbutto(padpf2)
      call qbutto(padpf3)
      call qbutto(padpf4)
      call qbutto(pad1)
      call qbutto(pad2)
      call qbutto(pad3)
      call qbutto(pad4)
      call qbutto(pad5)
      call qbutto(pad6)
      call qbutto(pad7)
      call qbutto(pad8)
      call qbutto(pad9)
      call qbutto(pad0)
      call qbutto(padent)
      call qbutto(rkey)
      call qbutto(tkey)
      call qbutto(onekey)
      call qbutto(twokey)
      call qbutto(threek)
      call qbutto(fourke)
      call qbutto(fiveke)
      call qbutto(sixkey)
      call qbutto(righta)
      call qbutto(leftar)

c     Define a dotted line style.

      call deflin(i2(1),i2(18568))

      return
      end
      subroutine inpath(lunit,initfl,idseam,values,endpat)

c     Generate a joint interpolated inter-seam trajectory, and test it
c     for interference.  A path is defined by a set of defined positions
c     which are visited in order.  The path between two positions is
c     joint-interpolated.  Two adjacent positions define the endpoints of
c     a path segment.

c     Parameters:     lunit-  a logic unit number
c                     initfl- .true. when a path is started.
c                     idseam- id of the weld immediately following the path.
c                     values- the joint values.
c                     endpat- .true. at end of path

c     Subroutines called: none

c     Called from: main

c     Variables:
c             nchain- number of kinematic chains
c             nrdof-  number of degrees of freedom
c             incr,decr- .true. to increase or decrease step size
c             values- joint values for a position
c             val2-   joint values for adjacent position
c             mincha- minimum allowable joint changes
c             delta-  joint changes between adjacent positions
c             diff-   joint increments for interpolated motion
c             time3-  elapsed time of current path
c             time4-  elapsed time of all inter-weld paths
c             time5-  total elapsed time of program

c     Saved variables:
c             isave-  the position from the previous loop
c             index1- the current position
c             index2- the current number of steps along the joint intepolated
c                     move
c             nend-   the final position along the path
c             nmax-   the number of increments between two positions
c             scale-  a scaling factor
c             isteps- number of increments per step
c
c

c*****************************************************************************

      character*20 itable,ilocat
      integer nrdof(3)
      logical initfl,endpat
      logical incr,decr
      real values(10,3)
      real mincha(10,3)
      real delta(10,3),diff(10,3),val2(10,3)
      integer recl(20)

      save isave,index1,index2,nend,nmax,scale,isteps,diff,tdiff
```

```
        common /nchain/nchain
        common /nrdof/nrdof
        common /itable/itable
        common /ilocat/ilocat
        common /mincha/mincha
        common /incr/incr
        common /decr/decr
        common /reclen/recl
        common /time/time1,time2,time3,time4,time5

c       Initialize at the start of the path.

        if (initfl) then
           open(unit=lunit,file=itable,form='unformatted',access='direct',
     *          status='old',recl=recl(7))
           read(lunit,rec=idseam) nstart,nend
           close(lunit)
           isteps=1
           scale=isteps
           index1=nstart
           index2=0
           isave=0
           time3=0.
           endpat=.false.

        else

c       Process the next segment of the current inter-weld path.

        if (isave.ne.index1) then

           isave=index1

c          Read the initial and final positions defining the segment
c          and read the time interval.

           open(unit=lunit,file=ilocat,form='unformatted',access='direct',
     *          status='old',recl=recl(8))
           read(lunit,rec=index1) ((values(j,i),j=1,nrdof(i)),i=1,nchain),timint
           read(lunit,rec=index1+1) ((val2(j,i),j=1,nrdof(i)),i=1,nchain),time
           close(lunit)

c          Calculate the minimum number of steps which ensure no joint change
c          limits are exceeded.

           nmax=0

           do 15 ind1=1,nchain
              do 20 ind2=1,nrdof(ind1)
                 delta(ind2,ind1)=val2(ind2,ind1)-values(ind2,ind1)
                 numb=int(abs(delta(ind2,ind1))/mincha(ind2,ind1))
                 if (numb.gt.ntemp) then
                    ntemp=numb
                 endif
```

```
20          continue
15        continue


c         Convert the number of steps to a power of 2.

          nmax=1
100       continue
          if (nmax.ge.ntemp) then
             goto 200
          else
             nmax=nmax*2
             goto 100
          endif
200       continue


c         Calculate the joint variable changes and time changes.

          do 25 ind1=1,nchain
             do 30 ind2=1,nrdof(ind1)
                diff(ind2,ind1)=delta(ind2,ind1)/nmax
                tdiff=time/nmax
30           continue
25        continue

       endif


c      Increase or decrease the step size by a factor of 2.

       if (incr) then
          incr=.false.
          if (isteps.lt.nmax) then
             scale=2.*scale
             isteps=2*isteps
          endif
       elseif (decr) then
          decr=.false.
          if (isteps.gt.1) then
             scale=scale/2.
             isteps=isteps/2
           endif
       endif


c         Test if the next step will move past the final position.  If so,
c         reduce the step size and try again.

          index2=index2+isteps

          if (index2.gt.nmax) then
             decr=.true.
             incr=.false.
             index2=index2-isteps
          endif
```

```
c          Increase all the joint values by one step.

           if (.not.decr) then
              do 40 ind1=1,nchain
                 do 45 ind2=1,nrdof(ind1)
                    values(ind2,ind1)=values(ind2,ind1)+scale*diff(ind2,ind1)
45               continue
40            continue

c             Increase the elapsed time.

              time3=time3+scale*tdiff
              time4=time4+scale*tdiff
              time5=time5+scale*tdiff
           endif

c          Check if a segment or path end is reached.

           if (index2.eq.nmax) then
              index2=0
              index1=index1+1
              if (index1.ge.nend) then
                 time3=0.
                 endpat=.true.
              endif
           endif

        endif

        return
        end
        subroutine input(icount)

c       Accept interactive user input and set flags.

c       Parameters:      icount- the number of welded seams.

c       Subroutines called: drtext

c       Graphics calls: ringbe

c       Called from: main

c       Common blocks modified: flags1,flags2,flags3,reset,start,rmweld,
c                               shord,mkpath,incrm1,incrm2,direct,edit,jmove,
c                               pquit,chmode

c       Files read,written: none

c       DEFINITION OF VARIABLES:

c       add-    .true. to add new position in 'edit' mode
c       boldsm- .true. to enable CAD seam highlighting
c       chmode- .true. to change mouse button functions
c       decr_   .true. for decrease
c       defwel- .true. to enable interactive weld path generation
c       direv-  .true. to reverse the direction of a user defined seam
c       edit-   .true. for inter-weld editing mode
c       forwrd- .true. for forward motion
c       incr-   .true. for increase
c       incrm1- .true. to jump to next inter-weld position
c       incrm2- .true. to jump to next inter-weld path with interference
c       intset- .true. to automatically save interfering paths during program
c               replay
c       jmove-  boolean array enabling motion selectively by joint
c       keyon-  .true. for interactive seam definition
c       makep-  flag used for inter-weld path definition
c       menu1-  menu selection flag
c       mkpath- flag used for inter-weld path definition
c       mov-    .true. to move a position in 'edit' mode
c       mrobot- .true. to enable robot motion in 'edit' mode
c       mtable- .true. to enable positioner motion in 'edit' mode
c       mvtabl- .true. for synchronous positioner motion
c       pickfl- .true. when picking mode is active
c       pquit-  .true. to stop replay before it is complete
c       rem-    .true. to remove a position in 'edit' mode
c       remove- .true. to remove a user-defined seam
c       replay- .true. to replay the robot program
c       reset-  .true. to remove all welds, then to remove all defined seams.
c       revers- .true. for reverse motion
c       rmweld- .true. to remove a weld
c       shdist- .true. to calculate exact separating distances
c       shord-  .true. to show the order of the welds
c       skip-   .true. to skip to the start of the next stage during replay
c       solid-  .true. for shaded, hidden-surface removed graphics
c       start-  .true. if the graphical image is to be reconstructed
c       step-   .true. for single step mode
```

```
c       tesint- .true. to test for intersection
c       tinit,tfinal-  .true. to select initial and final table positions
c       tjntl,tjnt2,tjnt3- flags to enable selective user control of positioner
c                          joints
c
c*********************************************************************************
$include /usr/include/fgl.h
$include /usr/include/fdevice.h

        logical keyon,replay,mvtabl,shdist,pickfl,solid,shrobt
        logical tinit,tfinal,start
        logical reset,remove,direv
        logical shift
        logical rmweld
        logical shord
        logical mkpath
        logical makep
        logical incrml
        logical incrm2
        logical forwrd,revers
        logical edit,add,rem,mov
        logical jmove(10,2)
        logical mrobot,mtable
        logical pquit
        logical stopfl
        logical chmode
        logical defwel
        logical menul
        logical incr,decr
        integer*2 data
        integer icount
        logical tesint
        logical intset
        logical step
        logical skip
        logical boldsm
        logical tjntl,tjnt2,tjnt3

        common /flagsl/keyon,replay,shdist,pickfl,solid,shrobt
        common /flags2/mvtabl,tinit,tfinal
        common /flags3/remove,direv
        common /reset/reset
        common /start/start
        common /rmweld/rmweld
        common /shord/shord
        common /mkpath/mkpath
        common /makep/makep
        common /incrml/incrml
        common /incrm2/incrm2
        common /direct/forwrd,revers
        common /edit/edit,add,rem,mov
        common /jmove/jmove
        common /pquit/pquit
        common /chmode/chmode
        common /defwel/defwel
        common /menul/menul
        common /incr/incr
        common /decr/decr
        common /tesint/tesint
        common /intset/intset
        common /step/step
        common /skip/skip
        common /tjnts/tjntl,tjnt2,tjnt3
        common /boldsm/boldsm

        save shift,mrobot,mtable


c       Test if a button has been hit.  Note that an event is queued twice for
c       each button, once when it is pressed and once when it is released.
c       Set a flag only when the button is pressed (i.e. its value is not 0).
c       Once a button has been identified and the appropriate flag set, set a
c       stop flag 'stopfl' to prevent any further actions.

        if (qtest().ne.0) then
           ibut=qread(posit)
           if (posit.ne.0) then

           stopfl=.false.


c       Change mouse button functions.

        if (ibut.eq.padpf2) then
           chmode=.not.chmode
           stopfl=.true.
           call ringbe

c       Reset.

        elseif (ibut.eq.padpf3) then
           reset=.true.
           edit=.false.
           mkpath=.false.
           makep=.false.
           menul=.true.
           stopfl=.true.
           call ringbe
           call drtext('RESET',' ')


c       Enable replay of program.

        elseif (ibut.eq.padpf4) then
           if (.not.replay) then
              replay=.true.
              edit=.false.
              mkpath=.false.
              makep=.false.
              menul=.false.
              defwel=.false.
```

```
               call ringbe
               call drtext('REPLAY PROGRAM ',' ')
            else
               pquit=.true.
               call ringbe
               call drtext('CANCELLED ',' ')
            endif
            stopfl=.true.

c        Selectively activate positioner joints for motion.

         elseif (ibut.eq.onekey) then
            if (.not.edit) then
               tjnt1=.true.
               tjnt2=.false.
               tjnt3=.false.
               stopfl=.true.
               call ringbe
               call drtext('POSITIONER JOINT 1 ACTIVATED',' ')
            endif

         elseif (ibut.eq.twokey) then
            if (.not.edit) then
               tjnt1=.false.
               tjnt2=.true.
               tjnt3=.false.
               stopfl=.true.
               call ringbe
               call drtext('POSITIONER JOINT 2 ACTIVATED',' ')
            endif

         elseif (ibut.eq.threek) then
            if (.not.edit) then
               tjnt1=.false.
               tjnt2=.false.
               tjnt3=.true.
               stopfl=.true.
               call ringbe
               call drtext('POSITIONER JOINT 3 ACTIVATED',' ')
            endif


c        Shift function.

         elseif (ibut.eq.padent) then
            shift=.not.shift
            stopfl=.true.

c        Enable picking mode.

         elseif (ibut.eq.mousel) then
            pickfl=.true.
            stopfl=.true.
            call ringbe
```

```
c       Switch between solid and wireframe depiction.

        elseif (shift.and..not.stopfl) then
        if (ibut.eq.pad1) then
           solid=.not.solid
           start=.true.
           shift=.false.
           stopfl=.true.
           call ringbe

c       Make the seams bold for visibility.

        elseif (ibut.eq.pad2) then
           boldsm=.not.boldsm
           shift=.false.
           stopfl=.true.
           call ringbe

c       Calculate and show exact separating distances.

        elseif (ibut.eq.pad3) then
           shdist=.not.shdist
           shift=.false.
           stopfl=.true.
           call ringbe

        elseif (ibut.eq.pad4) then
           tesint=.not.tesint
           shift=.false.
           stopfl=.true.
           call ringbe
           if (tesint) then
              call drtext('INTERFERENCE TESTING ON',' ')
           else
              call drtext('INTERFERENCE TESTING OFF',' ')
           endif

        elseif (ibut.eq.pad5) then
           intset=.not.intset
           shift=.false.
           stopfl=.true.
           call ringbe
           if (intset) then
              call drtext('PROCESS NEXT INTERFERING PATH',' ')
           else
              call drtext('PICK PATH TO PROCESS',' ')
           endif

        endif
        endif


c       Select mode.

        if (menul.and..not.stopfl) then
```

```
c       Enter seam definition mode.

        if (ibut.eq.pad1) then
           keyon=.true.
           stopfl=.true.
           call ringbe
           call drtext('DEFINE SEAMS',' ')

c       Enter weld definition mode.

        elseif (ibut.eq.pad2) then
           defwel=.true.
           stopfl=.true.
           menul=.false.

           call ringbe
           call drtext('DEFINE WELDS',' ')

c       Enter inter-weld path mode.

        elseif (ibut.eq.pad3) then
           if (icount.gt.0) then
              mkpath=.true.
              makep=.false.
              menul=.false.
              call ringbe
              if (intset) then
                 call drtext('INTER-WELD PATH- AUTO MODE',' ')
              else
                 call drtext('INTER-WELD PATH MODE- PICK WELD',' ')
              endif
           else
              call ringbe
              call drtext('THERE ARE NO WELDS',' ')
           endif
           stopfl=.true.


        endif
        endif



c       Inter-seam path editing mode.
c       Once a change is made, external routines set edit=.false.

        if (edit.and..not.stopfl) then

c       Exit without making a change.

        if (ibut.eq.0) then
           edit=.false.
           stopfl=.true.
           call ringbe
           call drtext('EXIT',' ')
```

```
c     Move robot.

      elseif (ibut.eq.rkey) then
         mrobot=.true.
         mtable=.false.
         stopfl=.true.
         call ringbe
         call drtext('MOVE ROBOT JOINTS',' ')

c     Move positioning table.

      elseif (ibut.eq.tkey) then
         mtable=.true.
         mrobot=.false.
         stopfl=.true.
         call ringbe
         call drtext('MOVE POSITIONING TABLE JOINTS',' ')

c     Add a position to the path.

      elseif (ibut.eq.pad1) then
         add=.true.
         stopfl=.true.
         call ringbe
         call drtext('ADD A NEW POSITION',' ')

c     Remove a position from the path.

      elseif (ibut.eq.pad2) then
         rem=.true.
         stopfl=.true.
         call ringbe
         call drtext('REMOVE A POSITION',' ')

c     Move a position.

      elseif (ibut.eq.pad3) then
         mov=.true.
         stopfl=.true.
         call ringbe
         call drtext('MOVE A POSITION',' ')

c     Single step mode.

      elseif (ibut.eq.pad6) then
         step=.not.step
         stopfl=.true.
         call ringbe
         if (step) then
            call drtext('SINGLE STEP MODE',' ')
         else
            call drtext('CONTINUOUS MODE',' ')
         endif


c     Move robot and positioner joints.
```

```
         elseif (mrobot) then
            if (ibut.eq.onekey) then
               jmove(1,1)=.not.jmove(1,1)
               stopfl=.true.
            elseif (ibut.eq.twokey) then
               jmove(2,1)=.not.jmove(2,1)
               stopfl=.true.
            elseif (ibut.eq.threek) then
               jmove(3,1)=.not.jmove(3,1)
               stopfl=.true.
            elseif (ibut.eq.fourke) then
               jmove(4,1)=.not.jmove(4,1)
               stopfl=.true.
            elseif (ibut.eq.fiveke) then
               jmove(5,1)=.not.jmove(5,1)
               stopfl=.true.
            elseif (ibut.eq.sixkey) then
               jmove(6,1)=.not.jmove(6,1)
               stopfl=.true.
            endif

         elseif (mtable) then

            if (ibut.eq.onekey) then
               jmove(1,2)=.not.jmove(1,2)
               stopfl=.true.
            elseif (ibut.eq.twokey) then
               jmove(2,2)=.not.jmove(2,2)
               stopfl=.true.
            elseif (ibut.eq.threek) then
               jmove(3,2)=.not.jmove(3,2)
               stopfl=.true.
            elseif (ibut.eq.fourke) then
               jmove(4,2)=.not.jmove(4,2)
               stopfl=.true.
            elseif (ibut.eq.fiveke) then
               jmove(5,2)=.not.jmove(5,2)
               stopfl=.true.
            elseif (ibut.eq.sixkey) then
               jmove(6,2)=.not.jmove(6,2)
               stopfl=.true.
            endif

         endif
         endif


c        Move in forward or reverse direction.

         if (makep.or.mkpath) then
            if (ibut.eq.righta) then
               forwrd=.not.forwrd
               revers=.false.
               stopfl=.true.
            call ringbe
            if (forwrd) then
               call drtext('MOVE FORWARD ALONG JOINT INTERPOLATED PATH',' ')
            else
               call drtext('STOP',' ')
            endif
         elseif (ibut.eq.leftar) then
            revers=.not.revers
            forwrd=.false.
            stopfl=.true.
            call ringbe
            if (revers) then
               call drtext('MOVE BACKWARD ALONG JOINT INTERPOLATED PATH',' ')
            else
               call drtext('STOP',' ')
            endif
         endif
      endif


c     Weld definition mode.

      if (defwel.and..not.stopfl) then

c     Choose initial and final positions of a moving table.

      if (ibut.eq.pad7) then
            mvtabl=.not.mvtabl
            stopfl=.true.
            call ringbe

      elseif (ibut.eq.pad8) then
            if (mvtabl) then
               tinit=.true.
            endif
            stopfl=.true.
            call ringbe

      elseif (ibut.eq.pad9) then
            if (mvtabl) then
               tfinal=.true.
            endif
            stopfl=.true.
            call ringbe

c     Exit this mode.

      elseif (ibut.eq.pad0) then
         defwel=.false.
         edit=.false.
         makep=.false.
         mkpath=.false.
         stopfl=.true.
         call ringbe
```

```
            menul=.true.
            call drtext('EXIT',' ')

c        Remove a previously defined seam.

         elseif (ibut.eq.pad1) then
            remove=.true.
            shift=.false.
            stopfl=.true.
            call ringbe
            call drtext('REMOVE NEXT SEAM PICKED',' ')

c        Reverse the previously defined seam direction.

         elseif (ibut.eq.pad2) then
            direv=.true.
            shift=.false.
            stopfl=.true.
            call ringbe
            call drtext('REVERSE DIRECTION OF NEXT SEAM PICKED',' ')

c        Unweld a previously welded seam.

         elseif (ibut.eq.pad3) then
            rmweld=.true.
            shift=.false.
            stopfl=.true.
            call ringbe
            call drtext('UNWELD NEXT SEAM PICKED',' ')

c        Show the order of the welded seams.

         elseif (ibut.eq.pad4) then
            shord=.not.shord
            shift=.false.
            stopfl=.true.
            call ringbe
            call drtext('SHOW THE WELDING ORDER',' ')

         endif
         endif


c        Define an inter-seam path.

c        Inter-weld path mode.

         if (makep.or.mkpath) then

c        Exit this mode.

         if (ibut.eq.pad0) then
            mkpath=.false.
            makep=.false.
            edit=.false.
```

```
           defwel=.false.
           incr=.false.
           decr=.false.
           stopfl=.true.
           call ringbe
           menul=.true.
           call drtext('EXIT',' ')
c       Process the next interfering path.
        elseif (ibut.eq.pad7) then
           incrm2=.true.
           stopfl=.true.
           call ringbe

        endif
        endif

        if (makep.and..not.stopfl) then
c       Increment defined robot positions.
        if (ibut.eq.pad1) then
              incrm1=.true.
           stopfl=.true.
           call ringbe
c       Interpolate intermediate robot positions.
c       Enable inter-seam path editing mode.
        elseif (ibut.eq.pad3) then
              edit=.true.
                 call drtext('INTER-WELD PATH EDITING MODE',' ')
           stopfl=.true.
           call ringbe
c       Increase step size.
        elseif (ibut.eq.pad4) then
           incr=.true.
           call drtext('INCREASE STEP SIZE',' ')
           stopfl=.true.
           call ringbe
c       Decrease step size.
        elseif (ibut.eq.pad5) then
           decr=.true.
           call drtext('DECREASE STEP SIZE',' ')
           stopfl=.true.
           call ringbe
c       Single step motion mode.
        elseif (ibut.eq.pad6) then
```

```
            step=.not.step
            stopfl=.true.
            call ringbe
            if (step) then
               call drtext('SINGLE STEP MODE',' ')
            else
               call drtext('CONTINUOUS MODE',' ')
            endif
         endif

         elseif (replay.and..not.stopfl) then

             if (ibut.eq.padl) then
                skip=.true.
                stopfl=.true.
                call ringbe
                call drtext('SKIP NEXT PART OF PROGRAM',' ')
             elseif (ibut.eq.pad4) then
                incr=.true.
                stopfl=.true.
                call ringbe
                call drtext('DOUBLE STEP SIZE',' ')
             elseif (ibut.eq.pad5) then
                decr=.true.
                stopfl=.true.
                call ringbe
                call drtext('HALVE STEP SIZE',' ')
             endif
         endif


         endif
         endif


         return
         end
```

```
      subroutine intest(angles,tabpos,istat)

c     Test if the proposed kinematically feasible path results in
c     interference. The sequence of robot joint angles are contained in a
c     file.  Each robot position is tested for interference.
c     The robot positions are shown on the screen as they are tested.

c     SUBROUTINES CALLED:
c             displl,drwire,plist,trandb

c     Called from: worpos

c     PARAMETERS:
c             angles- the joint angles of the robot
c             tabpos- the joint values for the positioning table
c             istat-  .false. if no interference
c                     .true. if interference detected

c     GRAPHICS ROUTINES:
c             callob,closeo,color,frontb,gflush,makeob,polarv,popmat,pushma,
c             viewpo,window

c     Common blocks modified: none

c     Variables:
c
c             colist- a list of parts which interfere
c             ftran1,ftran2-  transformations from world coords to torch and
c                             workpiece coords
c             grobjs- a list of graphical objects for seams
c             part-   a list of parts in the workstation
c             seamob,boldob,endob- graphical object ids for CAD reference
c                                  seam
c             values- the robot and positioner joint values
c************************************************************************

      real angles(*)
      real tabpos(*)
      integer part(100),number
      integer grobjs(100),nseq
      integer colist(100),icolst
      real values(10,3)
      real winsiz(6)
      real vdist
      integer*2 azim,inc,twist
      integer*2 left,right,bottom,top
      logical istat
      real ftran1(4,4),ftran2(4,4)
      integer seamob,boldob,endobj

      common /parts/part,number
      common /viewpo/left,right,bottom,top
      common /winsiz/winsiz
      common /view/vdist,azim,inc,twist
      common /nseams/nseams
      common /ftrans/ftranl,ftran2
      common /grobjs/grobjs
      common /nseq/nseq
      common /seamob/seamob,boldob,endobj
      common /values/values

      istat=.false.
      dmin=100000.

      call pushma
      call pushvi

      call frontb(.false.)
      call viewpo(left,right,bottom,top)
      call window(winsiz(1),winsiz(2),winsiz(3),winsiz(4),winsiz(5),winsiz(6))

      call pushvi
      call pushma

      call color(i2(0))
      call clear
      call color(i2(11))
      call drscrn
      call color(i2(15))

c     Read the joint angles for each robot position and test for interference.

      do 2 i=1,6
         values(i,1)=angles(i)
         values(i,2)=tabpos(i)
 2    continue
      call trandb(1,values)
      call plist(0,part,number,colist,icolst,dist)
      if (dist.lt.dmin) then
         dmin=dist
      endif

c     Change the colour if interference was detected.

      if (dmin.lt.0.) then
         istat=.true.
         call color(i2(12))
      endif

c     Draw the scene in wireframe.

      call drwire(0,part,number)

c     Draw the interfering parts in another colour.

      if (istat) then
         call color(i2(13))
         call drwire(0,colist,icolst)
      endif

c     Draw the seams.
```

```
        call polarv(vdist,azim,inc,twist)
        call multma(ftran2)
        call color(i2(12))
        call callob(seamob)
        do 20 i=1,nseq
           call callob(grobjs(i))
20      continue

c       Draw the information display.

        call displl
        call gflush

        call swapbu

        call popvie
        call popmat
        call color(i2(0))
        call clear

        call popvie
        call popmat

        return
        end
```

```
      SUBROUTINE JOINTS(H06,RTHETA,IFLAG)
C
C     THIS SUBROUTINE CALCULATES THE JOINT ANGLES OF A 6 DEGREE OF
C     FREEDOM MANIPULATOR GIVEN THE DESIRED LOCATION AND ORIENTATION
C     OF THE GRIPPER. THE MANIPULATOR TO BE SOLVED MUST BE
C     GEOMETRICALLY IDENTICAL TO A UNIMATION PUMA 560, WITH
C     THE JOINT DIMENSIONS BEING SPECIFIABLE.
C
C     INPUTS: H06(4,4)-  HOMOGENEOUS TRANSFORMATION MATRIX FROM THE
C                        SIXTH JOINT FRAME 'F6' TO THE ROBOT BASE
C                        WORLD FRAME 'F0'. 'H06' IS OF THE FORM
C                        [n,o,a,p], WHERE n,o,a ARE UNIT VECTORS
C                        DESCRIBING THE ORIENTATION OF FRAME 'F6',
C                        AND p IS THE POSITION VECTOR FOR THE ORIGIN
C                        OF 'F6' RELATIVE TO 'F0'.
C
C     OUTPUTS:  RTHETA:  A REAL ARRAY CONTAINING THE JOINT ANGLES IN
C                          DEGREES.
C
C               IFLAG:     THIS FLAG IS SET TO 1 IF THE ROBOT WORK
C                          ENVELOPE IS EXCEEDED, OTHERWISE IT IS 0.
C
C     COMMON: /CONFIG/ICONF(6)-INTEGER ARRAY DESCRIBING DESIRED
C                          CONFIGURATION OF EACH JOINT, WHEN MORE THAN
C                          ONE CONFIGURATION IS POSSIBLE.
C
C               POSSIBLE CONFIGURATIONS:
C
C               0 0 0 0 0 0 - L A NF
C               0 0 0 1 0 0 - L A F
C               0 0 1 0 0 0 - L B NF
C               0 0 1 1 0 0 - L B F
C               1 1 1 0 0 0 - R A NF
C               1 1 1 1 0 0 - R A F
C               1 1 0 0 0 0 - R B NF
C               1 1 0 1 0 0 - R B F
C
C             WHERE-  R=RIGHT SHOULDER CONFIG.
C                     L=LEFT SHOULDER CONFIG.
C                     A=ELBOW ABOVE HAND
C                     B=ELBOW BELOW HAND
C                     NF=WRIST NOT FLIPPED
C                     F=WRIST FLIPPED
C
C             NOTE:   ICONF(5-6) NOT USED
C
C
C     FILES:    NONE
C
c     Called from: worpos
C
C     SUBROUTINES CALLED:    NONE
C
C*****************************************************************************
C
      REAL H06(4,4),RTHETA(*),ALPHA(10,3),LEN(10,3),DIST(10,3),
     *  THETA(10,3)
        REAL X,Y,TEMP1,TEMP2,TEMP3,TEMP4,TEMP5,TEMP6,H6G(4,4),
     *  INH00(4,4),H00(4,4),H06(4,4),Q(6),INH6A(4,4)
        INTEGER ICONF(6),IFLAG
        INTEGER NRDOF(3),LINK(10,3),JTYPE(10,3)
C
        COMMON /INVERS/INH00,INH6A
        COMMON /ROBOT/LINK,JTYPE,ALPHA,LEN,THETA,DIST
        common /nrdof/nrdof
        COMMON /CONFIG/ICONF
C
C
C*****  INITIALIZE 'IFLAG'
C
        IFLAG=0
C
C*****  CALCULATE COMPONENTS OF VECTOR 'Q(3)'
C
        DO 20 I=1,3
           Q(I)=H06(I,4)-DIST(6,1)*H06(I,3)
20      CONTINUE
        Q(4)=1.
C
C*****  FIND RTHETA(1) **************************************************
C
        TEST=Q(1)**2+Q(2)**2-DIST(2,1)**2
C
C*****  IF(ENVELOPE EXCEEDED) SET FLAG AND RETURN
C
        IF(TEST.LT.0.) GO TO 850
C
        TEMP=SQRT(Q(1)**2+Q(2)**2-DIST(2,1)**2)
        T1=Q(2)*TEMP
        T2=Q(1)*TEMP
C
C*****  CHOOSE SOLUTION DEPENDING ON ROBOT CONFIGURATION.
C
        IF (ICONF(1).GT.1) GO TO 900
        IF (ICONF(1).NE.1) GO TO 100
           T1=-T1
           T2=-T2
100     CONTINUE
C
        Y=T1-DIST(2,1)*Q(1)
        X=T2+DIST(2,1)*Q(2)
C
        RTHETA(1)=ATAN2(Y,X)
C
C*****  FIND RTHETA(3) **************************************************
C
        Y=(Q(1)**2+Q(2)**2+Q(3)**2-DIST(4,1)**2-LEN(2,1)**2-DIST(2,1)**2-
     *  LEN(3,1)**2)/(2.*LEN(2,1))
        RSQ=LEN(3,1)**2+DIST(4,1)**2
        TEST2=RSQ-Y**2
C
C*****  IF(ENVELOPE EXCEEDED) SET FLAG AND RETURN
```

```
C
        IF(TEST2.LT.0.) GO TO 850
C
        X=SQRT(TEST2)
        THETA1=ATAN2(LEN(3,1),-DIST(4,1))
C
C*****  CHOOSE SOLUTION DEPENDING ON CONFIGURATION.
C
        IF (ICONF(3).GT.1) GO TO 900
        IF (ICONF(3).EQ.1) X=-X
C
        RTHETA(3)=THETA1-ATAN2(Y,X)
C
C*****  FIND RTHETA(2) *************************************************
C
        C3=COS(RTHETA(3))
        S3=SIN(RTHETA(3))
        TEMP1=DIST(4,1)*C3-LEN(3,1)*S3
        TEMP2=DIST(4,1)*S3+LEN(3,1)*C3+LEN(2,1)
        THETA1=ATAN2(TEMP1,TEMP2)
        RSQ=TEMP1**2+TEMP2**2
        TEST3=RSQ-Q(3)**2
C
C*****  IF(ENVELOPE EXCEEDED) SET FLAG AND RETURN
C
        IF(TEST3.LT.0.) GO TO 850
C
        Y=Q(3)
        X=SQRT(TEST3)
C
C*****  CHOOSE SOLUTION DEPENDING ON CONFIGURATION.
C
        IF(ICONF(2).NE.0) X=-X
C
        RTHETA(2)=THETA1-ATAN2(Y,X)
C
C*****  FIND RTHETA(4) *************************************************
C
        Y=COS(RTHETA(1))*H06(2,3)-SIN(RTHETA(1))*H06(1,3)
        X=COS(RTHETA(1))*COS(RTHETA(2)+RTHETA(3))*H06(1,3)+
     *    SIN(RTHETA(1))*COS(RTHETA(2)+RTHETA(3))*H06(2,3)-
     *    SIN(RTHETA(2)+RTHETA(3))*H06(3,3)
C
C*****  CHOOSE SOLUTION DEPENDING ON CONFIGURATION.
C
        IF(ICONF(4).GT.1) GO TO 900
        IF(ICONF(4).NE.1) GO TO 101
           X=-X
           Y=-Y
101     CONTINUE
C
        RTHETA(4)=ATAN2(Y,X)
C
C*****  FIND RTHETA(5) *************************************************
C
        TEMP1=(COS(RTHETA(1))*COS(RTHETA(2)+RTHETA(3))*COS(RTHETA(4))-
     *        SIN(RTHETA(1))*SIN(RTHETA(4)))*H06(1,3)
        TEMP2=(SIN(RTHETA(1))*COS(RTHETA(2)+RTHETA(3))*COS(RTHETA(4))+
     *        COS(RTHETA(1))*SIN(RTHETA(4)))*H06(2,3)
        TEMP3=COS(RTHETA(4))*SIN(RTHETA(2)+RTHETA(3))*H06(3,3)
        TEMP4=COS(RTHETA(1))*SIN(RTHETA(2)+RTHETA(3))*H06(1,3)
        TEMP5=SIN(RTHETA(1))*SIN(RTHETA(2)+RTHETA(3))*H06(2,3)
        TEMP6=COS(RTHETA(2)+RTHETA(3))*H06(3,3)
C
        Y=TEMP1+TEMP2-TEMP3
        X=TEMP4+TEMP5+TEMP6
C
        RTHETA(5)=ATAN2(Y,X)
C
C*****  FIND RTHETA(6) ************************************************
C
        TEMP1=(-SIN(RTHETA(1))*COS(RTHETA(4))-COS(RTHETA(1))*COS(RTHETA(2)+
     *        RTHETA(3))*SIN(RTHETA(4)))*H06(1,1)
        TEMP2=(COS(RTHETA(1))*COS(RTHETA(4))-SIN(RTHETA(1))*COS(RTHETA(2)+
     *        RTHETA(3))*SIN(RTHETA(4)))*H06(2,1)
        TEMP3=SIN(RTHETA(4))*SIN(RTHETA(2)+RTHETA(3))*H06(3,1)
        TEMP4=(-SIN(RTHETA(1))*COS(RTHETA(4))-COS(RTHETA(1))*COS(RTHETA(2)+
     *        RTHETA(3))*SIN(RTHETA(4)))*H06(1,2)
        TEMP5=(COS(RTHETA(1))*COS(RTHETA(4))-SIN(RTHETA(1))*COS(RTHETA(2)+
     *        RTHETA(3))*SIN(RTHETA(4)))*H06(2,2)
        TEMP6=SIN(RTHETA(4))*SIN(RTHETA(2)+RTHETA(3))*H06(3,2)
C
        Y=TEMP1+TEMP2+TEMP3
        X=TEMP4+TEMP5+TEMP6
C
        RTHETA(6)=ATAN2(Y,X)
C
C*****  CONVERT ANGLES INTO DEGREES, INSURE THAT -180.LE.THETA.LE.180.
C
        CONV=180./3.14159
        DO 30 I=1,6
           RTHETA(I)=RTHETA(I)*CONV
           IF(RTHETA(I).GT.180.) RTHETA(I)=RTHETA(I)-360.
           IF(RTHETA(I).LT.-180) RTHETA(I)=RTHETA(I)+360.
30      CONTINUE
C
        GO TO 800
C
C*****  POSITION OUTSIDE ENVELOPE, SET FLAG
C
850     CONTINUE
           IFLAG=1
        GO TO 800
C
C*****  ERROR MESSAGES  ***********************************************
C
900     WRITE(*,901)
901     FORMAT(' VALUE OF PARAMETER ICONF NOT 0 OR 1')
C
800     CONTINUE
C
        RETURN
```

```
      END
```

```
        SUBROUTINE LIMITS(RTHETA,IFLAG,IJOINT)
C
C       THIS SUBROUTINE CHECKS THE CALCULATED robot JOINT ANGLES 'THETA'
C       AGAINST THEIR PHYSICAL LIMITS.  IF ANY JOINT ANGLE IS
C       EXCEEDED, 'IFLAG' IS SET.
C
C
C       INPUTS: RTHETA- JOINT ANGLES IN DEGREES
C
C
C       OUTPUTS: IFLAG- 0-ANGLES WITHIN LIMITS
C                       1-JOINT LIMIT EXCEEDED
C
C                IJOINT- ARRAY WHICH IDENTIFIES JOINTS
C                          WHOSE LIMITS HAVE BEEN EXCEEDED.
C                       IE. 0 0 0 1 0 0 - JOINT 4 EXCEEDED
C
c       Called from:    worpos

c       Variables:

c               nrdof-  number of degrees of freedom
c               rmin,rmax-  minimum and maximum joint limits
C
C
C***************************************************************************
        REAL RTHETA(*),RMIN(10,3),RMAX(10,3)
        INTEGER IJOINT(*)
        integer nrdof(3)
C
        common /jlimit/rmin,rmax
        common /nrdof/nrdof
C
C
C*****  INITIALIZE 'IJOINT','IFLAG'
C
        DO 11 I=1,nrdof(1)
           IJOINT(I)=0
11      CONTINUE
C
        IFLAG=0
C
C*****  TEST EACH JOINT ANGLE 'THETA'AGAINST ITS SPECIFIED LIMITS
C*****  'RMIN','RMAX'.
C
        DO 20 I=1,nrdof(1)
C
           IF(RMIN(I,1).LE.RMAX(I,1)) GO TO 200
C
           IF(RTHETA(I).GE.RMIN(I,1).OR.RTHETA(I).LE.RMAX(I,1)) GO TO 100
              IFLAG=1
              IJOINT(I)=1
              GO TO 100
200        CONTINUE
C
           IF(RTHETA(I).GE.RMIN(I,1).AND.RTHETA(I).LE.RMAX(I,1)) GO TO 100
```

```
               IFLAG=1
              IJOINT(I)=1
100        CONTINUE
C
20      CONTINUE
C
        RETURN
        END
        subroutine lsort(ipick,icount,grobjs,sequen,nseq,seams)

c       Update and re-sort the seam and weld definition data structures.

c       Parameters:       ipick-  array of welded seams.
c                         icount- number of welded seams.
c                         grobjs- array of seam graphical object ids.
c                         sequen- array of pointers into seam table.
c                         nseq-   number of segments in a sequence comprising a
c                                 seam.
c                         seams-  array containing, for each segment, the
c                                 reference seam id, and the starting and ending
c                                 points.

c       Subroutines called: none

c       Called from: main

c*****************************************************************************

        integer ipick(*),icount,grobjs(*),sequen(2,*),nseq,seams(3,*)


        index=0
        iseam=0


        do 10 ind=1,nseq

           if (grobjs(ind).ne.0) then
              iseam=iseam+1
              istart=sequen(1,ind)
              iend=istart+sequen(2,ind)-1
              sequen(1,iseam)=index+1
              do 20 ind2=istart,iend
                 index=index+1
                 do 30 i=1,3
                    seams(i,index)=seams(i,ind2)
30               continue
20            continue
              sequen(2,iseam)=sequen(2,ind)

              grobjs(iseam)=grobjs(ind)

              do 40 ind3=1,icount
                 if (ipick(ind3).eq.ind) then
                    ipick(ind3)=iseam
                 endif
40            continue
           endif

10      continue

        nseq=iseam
```

```
      return
      end
```

```
c       This module contains general purpose mathematical routines. The
c       following subroutines are included:
c
c                 det2x2  det3x3  dotprd  matdet  matinv  nxprod
c                 sol3x3  solve   unit    vdiff   vlen    xprod
c
c*****************************************************************************

        subroutine vdiff(v2,v1,dif)

c       This subroutine performs the vector difference operation
c       dif=v2-v1.

        real v1(3),v2(3),dif(3)

        do 10 i=1,3
           dif(i)=v2(i)-v1(i)
10      continue

        return
        end

c*****************************************************************************

        subroutine det2x2(v1,v2,det)

c       This subroutine finds the determinant of of a 2x2 matrix of the
c       form [v1,v2].

        real v1(2),v2(2),det

        det=v1(1)*v2(2)-v1(2)*v2(1)

        return
        end

c*****************************************************************************

        subroutine det3x3(v1,v2,v3,det)

c       Find the determinant of a 3x3 matrix of the form [v1,v2,v3].

        real v1(3),v2(3),v3(3),det

        det=v1(1)*v2(2)*v3(3)+v1(2)*v2(3)*v3(1)+v1(3)*v2(1)*v3(2)-
     *      v1(3)*v2(2)*v3(1)-v1(2)*v2(1)*v3(3)-v1(1)*v2(3)*v3(2)

        return
        end

c*****************************************************************************
```

```
        subroutine dotprd(v1,v2,prod)

c       Find the dot product of two vectors.

        real v1(3),v2(3),prod

        prod=v1(1)*v2(1)+v1(2)*v2(2)+v1(3)*v2(3)

        return
        end

c*******************************************************************************

        subroutine matdet(rmat,det)

c       Find the determinant of a 3 x 3 matrix 'rmat'


        real rmat(3,3),det

        det=rmat(1,1)*(rmat(2,2)*rmat(3,3)-rmat(2,3)*rmat(3,2))-
     *      rmat(1,2)*(rmat(2,1)*rmat(3,3)-rmat(2,3)*rmat(3,1))+
     *      rmat(1,3)*(rmat(2,1)*rmat(3,2)-rmat(2,2)*rmat(3,1))

        return
        end

c*******************************************************************************

        subroutine sol3x3(rmat,const,y,isol)

c       This subroutine finds the solution of a set of linear equations of
c       the form:
c               rmat(3,3)*y(3)+const(3)=0.

c       isol   0-  solution exists
c              1-  no solution


        real rmat(3,3),const(3),y(3),temp(3,3)

        isol=0

c       Find the determinant of the matrix 'rmat' and check for singularity.

        call matdet(rmat,det)

        if (abs(det).gt.toler) then

c       Solve for 'y'.

           do 10 i=1,3
              do 20 j=1,3
```

```
                 k1=i+1
                 k2=i+2
                 if (k1.gt.3) k1=k1-3
                 if (k2.gt.3) k2=k2-3

                 temp(j,i)=-const(j)
                 temp(j,k1)=rmat(j,k1)
                 temp(j,k2)=rmat(j,k2)

20            continue

              call matdet(temp,dtemp)
              y(i)=dtemp/det

10         continue

        else

           isol=1

        endif

        return
        end

c*******************************************************************************

        subroutine solve(p1,q1,u1,u2,u,dist,lamb1,lamb2)

c       This subroutine solves for the parameters 'lamb1','lamb2'
c       defining the common normal points on two lines:
c
c       line 1-  r(3)=p1(3)+lamb1*u1(3)
c       line 2-  r(3)=p2(3)+lamb2*u2(3)

c       u(3) is the unit common normal of the two lines
c       dist is the length of u(3).


        real p1(3),q1(3),u1(3),u2(3),u(3),pint(3),m1(2),m2(2),m3(2)
        real c(3),qp(3)
        real lamb1,lamb2
        logical nostop,repeat

        toler=.01

        icount=0

c       find two equations which give a non-zero determinant for the
c       denominator.

        repeat=.true.
        do 20 i=1,3
           if (repeat) then
```

```
              icount=icount+1
              j=i+1
              k=i+2
              if (i.eq.3) j=1
              if (i.gt.1) k=k-3

              m1(1)=u1(i)
              m1(2)=u1(j)
              m2(1)=u2(i)
              m2(2)=u2(j)

              call det2x2(m1,m2,det)

              isave=i
              jsave=j
              ksave=k

              if (abs(det).gt.toler) repeat=.false.
           endif
20      continue

        if (icount.le.3) then

        nostop=.true.
        do 50 ind=1,2
           if (nostop) then

c          calculate the value of the right hand side of the equation

           call vdiff(q1,p1,qp)
           do 10 i=1,3
              c(i)=qp(i)-dist*u(i)
10         continue

           m3(1)=c(isave)
           m3(2)=c(jsave)

c          solve for the unknowns lamb1,lamb2

           call det2x2(m3,m2,temp1)
           call det2x2(m1,m3,temp2)

           lamb1=temp1/det
           lamb2=-temp2/det

c          test if this is a valid solution
c          if not, change the sign of 'dist' and solve again.

           test=lamb1*u1(ksave)-lamb2*u2(ksave)-c(ksave)
           if (abs(test).gt.toler) then
              dist=-dist
           else
              nostop=.false.
           endif
        endif
50      continue
```

```
      endif
      return
      end

c***********************************************************************************

      subroutine unit(vect,uvect)

c     Normalize a vector.


      real vect(3),uvect(3),length

      length=sqrt(vect(1)**2+vect(2)**2+vect(3)**2)

      do 10 i=1,3
         uvect(i)=vect(i)/length
10    continue

      return
      end


c***********************************************************************************

      subroutine vlen(vector,length)

c     Find the scalar length of a vector.


      real length,vector(3)

      length=sqrt(vector(1)**2+vector(2)**2+vector(3)**2)

      return
      end


c***********************************************************************************

      subroutine xprod(a,b,c)

c     Find the cross product c = a X b


      real a(3),b(3),c(3)

      c(1)=a(2)*b(3)-a(3)*b(2)
      c(2)=a(3)*b(1)-a(1)*b(3)
      c(3)=a(1)*b(2)-a(2)*b(1)

      return
      end

c*******************************************************************************
      SUBROUTINE MATINV(H,HINV)
C
C     THIS SUBROUTINE FINDS THE INVERSE 'HINV' OF A HOMOGENEOUS
C     TRANSFORMATION MATRIX 'H'.
C
C     INPUTS: H(4,4)- HOMOGENEOUS TRANSFORMATION MATRIX
C
C     OUTPUTS: HINV(4,4)- INVERSE OF 'H'

c     Called from: init
C
C*****************************************************************************
C
      REAL H(4,4),HINV(4,4)
C
      DO 1 I=1,3
         DO 2 J=1,3
            HINV(I,J)=H(J,I)
2        CONTINUE
1     CONTINUE
C
      HINV(1,4)=-(H(1,4)*H(1,1)+H(2,4)*H(2,1)+H(3,4)*H(3,1))
      HINV(2,4)=-(H(1,4)*H(1,2)+H(2,4)*H(2,2)+H(3,4)*H(3,2))
      HINV(3,4)=-(H(1,4)*H(1,3)+H(2,4)*H(2,3)+H(3,4)*H(3,3))
C
      HINV(4,1)=0.
      HINV(4,2)=0.
      HINV(4,3)=0.
      HINV(4,4)=1.
C
      RETURN
      END


c*******************************************************************************
      SUBROUTINE NXPROD(A,B,C)
C
C     THIS SUBROUTINE CALCULATES VECTOR C = A X B (CROSS PRODUCT).
C
C     INPUTS:
C             A(3)=FIRST VECTOR
C             B(3)=SECOND VECTOR
C
C     OUTPUT:
C             C(3)=NORMALIZED CROSS PRODUCT, A X B
C
C*********************************************************************
C
```

```
C
        REAL A(3), B(3), C(3), ABS
C
        C(1)=A(2)*B(3)-A(3)*B(2)
        C(2)=A(3)*B(1)-A(1)*B(3)
        C(3)=A(1)*B(2)-A(2)*B(1)
C
        ABS=SQRT(C(1)**2+C(2)**2+C(3)**2)
        IF(ABS.EQ.0.) GO TO 100
C
        C(1)=C(1)/ABS
        C(2)=C(2)/ABS
        C(3)=C(3)/ABS
C
100     CONTINUE
        RETURN
        END


c*******************************************************************************
        INTEGER FUNCTION I2(I)
C
C       THIS FUNCTION CONVERTS TYPE INTEGER TO TYPE INTEGER*2
c       for IRIS fortran graphics calls.
c       This function is machine dependent.
C
c*******************************************************************************
        INTEGER*2 M
        EQUIVALENCE (M,N)
        M=I
        I2=N
C
        RETURN
        END
```

```
      subroutine mkseam(keyon,files,nfile)

c     Compose a user-defined seam interactively using pick.

c     Parameters:     keyon-  .true. if in the seam definition mode
c                     files-  array of seam segments
c                     nfile-  the number of seam segments

c     Subroutines called:  drtext,evpick,picksm

c     Graphics calls: ringbe

c     Called from: mksequ

c     Parameters modified: keyon,files,nfile

c     Common blocks modified: seamob,nsave

c     Variables:

c             tags-   tag, first and last offsets of picked object
c             lastag- previous 'tags'
c             buffer- a buffer for picking

c     Saved variables:

c             fcount- count of segments contained in seam definition
c             miss-   count of unsuccessful picks during seam definition.
c                     Seam definition is terminated after two consecutive
c                     misses.
c             nlast-  the number of tags in the last pick, 'sfiles'
c             sfiles- previously picked segment
c*******************************************************************************

      integer tags(3,10),lastag(3,10),npick,nlast
      integer sfiles(3,10),files(3,10)
      integer nsave,seamob,boldob,endobj,buffer(300)

      integer ntags,istat
      logical pstat,keyon
      integer nfile
      integer fcount
      integer miss
      integer*2 deltax,deltay

      save miss,nlast,fcount,sfiles

      common /seamob/seamob,boldob,endobj
      common /nsave/nsave

c     Define a small picking window.

      deltax=3
      deltay=3
```

```
        call picksi(deltax,deltay)
c       Continue until two consecutive misses are detected.

        if (miss.ge.2) then
           keyon=.false.
           call ringbe
           call drtext('EXIT',' ')
        endif

        if (keyon) then

c          Pick and evaluate seam.

           call picksm(nsave,1,seamob,buffer,npick,pstat)
           if (pstat) then
           if (npick.gt.1) then
              miss=0
              call evpick(nsave,buffer,ntags,tags,istat)

c             Find a match between the current and previous segments.
c             In case of multiple matches, the first match is used.

              nmatch=0
              ind2=1
              do 10 ind1=1,nlast
100              continue
                    if (lastag(1,ind1).gt.tags(1,ind2).and.ind2.lt.ntags) then
                       ind2=ind2+1
                       goto 100
                    endif

                 if (lastag(1,ind1).eq.tags(1,ind2)) then
                    nmatch=nmatch+1
                    if (nmatch.eq.1) then
                       call ringbe
                       fcount=fcount+1
                       sfiles(1,fcount)=lastag(1,ind1)
c                Test for endpoints.

                       if (tags(2,ind2).eq.1) then
                          tags(3,ind2)=1
                       endif
                       sfiles(2,fcount)=lastag(3,ind1)
                       sfiles(3,fcount)=tags(3,ind2)

                    endif
                endif

10             continue

c              Update the previous seam.
```

```
              if (fcount.eq.0.or.nmatch.ge.1) then
                 nlast=ntags
                 do 15 j=1,ntags
                    do 20 i=1,3
                       lastag(i,j)=tags(i,j)
20                  continue
15               continue
              endif

           else
              miss=miss+1

           endif
           endif
        else
c
           Copy the seam data into the output array.
c
           do 30 i=1,fcount
              do 35 j=1,3
                 files(j,i)=sfiles(j,i)
35            continue
30         continue
           nfile=fcount
c
           Reset the local variables.
c
           fcount=0
           nlast=0
           miss=0
        endif

c
        Restore the original picking window
c
        deltax=10
        deltay=10

        call picksi(deltax,deltay)

        return
        end
```

```
      subroutine mksequ(keyon,nseq,sequen,seams,grobjs)

c     Generate a sequence of user defined seams.

c     Parameters:     keyon-  .true. if in seam definition mode
c                     nseq-   number of seams in a sequence
c                     seams-  table of segment ids and pointers to first
c                             and last segment location records
c                     sequen- table of user defined seams. Each record
c                             contains a pointer to the first segment and the
c                             number of segments in 'seams'.
c                     grobjs- array of graphical objects

c     Subroutines called: drsequ,mkseam

c     Called from: main

c     Parameters modified: keyon,nseq,sequen,seams,grobjs

c     Common blocks modified: update

c     Files read: 'cadlst'

c     Variables:

c             files-  array of user defined segments comprising a seam
c             nfiles- the number of segments in 'files'

c     Saved varibles:

c             next-   the next seam starting point

c*****************************************************************************

      integer files(3,10),nfiles,sequen(2,*),seams(3,*)

      character*20 cadlst
      integer grobjs(*)
      logical keyon
      logical update
      integer recl(20)

      common /cadlst/cadlst
      common /reclen/recl
      common /update/update

      save next

c     Define a seam.

      if (keyon) then
         call mkseam(keyon,files,nfile)
      endif

c     Offset the segment pointers and
c     add the new seam to the seam sequence.
```

```
        if (.not.keyon) then
           if (nseq.eq.0) next=1
           nseq=nseq+1
           sequen(1,nseq)=next
           sequen(2,nseq)=nfile

           open(unit=1,file=cadlst,form='unformatted',access='direct',
     *          status='old',recl=recl(2))
           icount=1
           do 10 ind=next,next+nfile-1
              read(1,rec=files(1,icount)) nstart,npoint
              files(2,icount)=nstart+files(2,icount)-1
              files(3,icount)=nstart+files(3,icount)-1
              do 15 j=1,3
                 seams(j,ind)=files(j,icount)
15            continue
              icount=icount+1
10         continue
           close(1)

           next=next+nfile


c          Generate graphical objects.

           call drsequ(files,nfile,iobj)
           grobjs(nseq)=iobj

           update=.true.

        endif

        return
        end
```

```
      subroutine movpos(lunit,index,values)

c     Relocate a position on an inter-seam path.

c     Parameters:       lunit-  logical unit number
c                       index-  location of a position in the positions table
c                       values- the new robot and table joint values
c                               for a position

c     Subroutines called: none

c     Called from: editpa

c     File 'ilocat' is modified.

c***************************************************************************


      character*20 itable,ilocat
      integer nrdof(3),savend
      real save1(10,3),save2(10,3)
      real values(10,3)
      integer recl(20)


      common /itable/itable
      common /ilocat/ilocat
      common /nrdof/nrdof
      common /reclen/recl



c     Modify the joint values of an existing position.

      open(unit=lunit,file=ilocat,form='unformatted',access='direct',
     *    status='old',recl=recl(8))

      write(lunit,rec=index) (values(i,1),i=1,nrdof(1)),
     *   (values(i,2),i=1,nrdof(2)),time

      close(lunit)

      return
      end
```

```
        subroutine mtable(values)
c       Calculate the transformation relating the positioning table reference
c       frame to the world frame
c       by performing a chain of transformations on each link.
c       The transformations are in a standard form, and are functions of
c       given joint variables.

c       PARAMETERS:
c               values- the values of the joint variables

c       SUBROUTINES CALLED: none

c       Called from: worpos

c       Common blocks modified: ftrans

c       GRAPHICS CALLS:
c               loadma,popmat,pushma,rotate,transl

c       Variables:
c               ftran2- transformation from table frame to world frame
c               hpre2-  fixed table base to world frame transformation
c               hpost2- fixed transformation from last link frame to table
c                       frame
c               jtype-  the joint type, revolute or prismatic
c               len,dist,alpha,theta- Denavit-Hartenberg link specifications
c               nrdof-  the number of degrees of freedom
c

c*******************************************************************************

        integer nrdof(3)
        integer jtype(10,3)
        integer link(10,3)
        real len(10,3),dist(10,3),alpha(10,3),theta(10,3)
        real values(10)
        integer*2 rot,iangle
        real ident(4,4)
        real hprel(4,4),hpostl(4,4),hpre2(4,4),hpost2(4,4)
        real ftranl(4,4),ftran2(4,4)
        real temp(4,4)

        common /robot/link,jtype,alpha,len,theta,dist
        common /nrdof/nrdof
        common /trans/hprel,hpostl,hpre2,hpost2
        common /ftrans/ftranl,ftran2
        common /ident/ident

        call pushma

c       Transform the part clusters constituting the links of the positioning
c       table. Each link is related to the
c       previous link by the Denavit-Hartenberg transformation. The net
```

```
c       transformation is the product of these transformations.

        ind1=2

c       Premultiply by a fixed transformation.

           call loadma(ident)
           call multma(hpre2)

c       Perform chain of link transformations.

           do 20 ind2=1,nrdof(ind1)
              if (jtype(ind2,ind1).eq.0) then

c          Revolute joint.

                 rot=ifix(values(ind2)*10.)
                 call rotate(rot,'z')
                 call transl(len(ind2,ind1),0,dist(ind2,ind1))
              else

c          Prismatic joint.

                 rot=ifix(theta(ind2,ind1))
                 rot=0
                 call rotate(rot,'z')
                 call transl(len(ind2,ind1),0,values(ind2))
              endif
              iangle=10*alpha(ind2,ind1)
              call rotate(iangle,'x')

20         continue


c          Postmultiply by a fixed transformation to get the positioning table
c          frame to world frame transformation 'ftran2'.

           call multma(hpost2)
           call getmat(ftran2)


        call popmat

        return
        end
        SUBROUTINE NOAP5(ISOLN)
C
C       THIS SUBROUTINE FINDS THE 5 DEGREE OF FREEDOM PUMA LOCATION
C       PARAMETERS X,Y,Z,O,A,T GIVEN THE TOOL CENTER LOCATION VECTOR
C       'P', AND THE TOOL AXIS ORIENTATION VECTOR 'A'.
C
C       INPUTS: ISOLN- FLAG SPECIFYING WHICH OF THE TWO 5 DEGREE OF
C                      FREEDOM SOLUTIONS TO FIND.
C                       0- SOLUTION 1
C                       1- SOLUTION 2
C
c       Called from: worpos
c
c       Subroutines called: nxprod
C
C       COMMON BLOCKS READ:     /ROBOT/- ROBOT PARAMETERS
C                               /CONFIG/- CURRENT ROBOT CONFIGURATION
C                               /ROBPOS/- TOOL FRAME, H0G=[n,o,a,p]
C
C       COMMON BLOCKS ALTERED:  /ROBPOS/- TOOL FRAME, H0G=[n,o,a,p]
C
c       Variables:
c               alpha,len,dist,theta- Denavit-Hartenberg link parameters
c               h0g-    gripper to base transformation
c               iconf-  robot configuration
c               hpost1- robot gripper to last link transformation
c
C*******************************************************************************
C
        REAL N(3),O(3),A(3),P(3)
        REAL H0G(4,4)
        REAL ALPHA(10,3),LEN(10,3),DIST(10,3),
     *  THETA(10,3)
        INTEGER ICONF(6)
        real hpre1(4,4),hpost1(4,4),hpre2(4,4),hpost2(4,4)
        integer link(10,3),jtype(10,3)
c
        COMMON /ROBOT/LINK,JTYPE,ALPHA,LEN,THETA,DIST
        common /trans/hpre1,hpost1,hpre2,hpost2
        COMMON /CONFIG/ICONF
        COMMON /ROBPOS/H0G
C
C*****  READ THE VALUES OF TOOL FRAME VECTORS 'A','P'.
C
        DO 30 I=1,3
           A(I)=H0G(I,3)
           P(I)=H0G(I,4)
30      CONTINUE
C
C*****  FIND THE TOOL OFFSET IN THE X DIRECTION RELATIVE TO THE TOOL
C*****  MOUNTING FLANGE.
C
        DT=HPOST1(1,4)
C
C*****  CALCULATE THETA1
C
```

```
        T1=P(1)-DT*A(1)
        T2=P(2)-DT*A(2)
        RSQ=T1**2+T2**2
        ATAN1=ATAN2(T2,T1)
        T3=SQRT(RSQ-DIST(2,1)**2)
        IF(ICONF(1).EQ.1) T3=-T3
        ATAN=ATAN2(DIST(2,1),T3)
        THET1=ATAN1-ATAN
C
C*****  CALCULATE N(1). THERE ARE TWO POSIBLE SOLUTIONS, SELECTED BY THE
C*****  VALUE OF 'ISOLN'.
C
        C1=A(2)**2+A(3)**2
        C2=C1*A(3)**2
        C3=C1**2*TAN(THET1)**2
        C4=2.*C1*TAN(THET1)*A(1)*A(2)
        C5=C1*(A(1)**2+A(3)**2)
        N(1)=SQRT(C2/(C3+C4+C5))
        IF(ISOLN.EQ.1) N(1)=-N(1)
C
C*****  CALCULATE N(2)
C
C       D1=N(1)*A(1)*A(2)
C       D2=SQRT(4.*D1**2-4.*(A(2)**2+A(3)**2)*N(1)**2*(A(1)**2*
C     * A(3)**2)-A(3)**2)
C       D3=2.*(A(2)**2+A(3)**2)
        N(2)=N(1)*TAN(THET1)
C
C*****  CALCULATE N(3)
C
        N(3)=SQRT(1.-N(1)**2-N(2)**2)
        DOT=ABS(N(1)*A(1)+N(2)*A(2)+N(3)*A(3))
        IF(DOT.GT..0001) N(3)=-N(3)
C
C
C*****  CALCULATE BASIS VECTOR 'O'
C
        CALL NXPROD(A,N,O)
C
C
C*****  WRITE BASIS VECTORS 'N','O' INTO 'HOG'.
C
        DO 200 I=1,3
           HOG(I,1)=N(I)
           HOG(I,2)=O(I)
200     CONTINUE
C
        RETURN
        END
```

```
      SUBROUTINE NORM(N1,N2,A)
C
C     THIS SUBROUTINE CALCULATES THE NORMAL VECTOR, 'A', WHICH
C     BISECTS THE TWO PARENT SURFACE NORMALS 'N1','N2'.
C
C     INPUTS:
C           N1(3),N2(3)=NORMAL VECTORS OF TWO SURFACES
C
C     OUTPUT:
C           A(3)=UNIT BISECTOR OF N1 AND N2 AT POINT I+1.

c     Called from: proces
c
C**********************************************************************
C
C
      REAL N1(3),N2(3),A(3),LX,LY,LZ,L
C
      D1=N1(1)+N2(1)
      D2=N1(2)+N2(2)
      D3=N1(3)+N2(3)
C
      LX=D1**2
      LY=D2**2
      LZ=D3**2
C
      L=SQRT(LX+LY+LZ)
C
      A(1)=D1/L
      A(2)=D2/L
      A(3)=D3/L
C
      RETURN
      END
```

```
        subroutine obsort(nparts,part,bbox,ptlist)

c       This subroutine does a preliminary sort of a set of polyhedra according
c       to maximum screen depth, and calculates the bounding boxes for each
c       polyhedron.

c       CALLED FROM:
c               psort
c
c       SUBROUTINES CALLED:
c               ajvrts
c               fplist
c               getcrd
c               getvrt
c               screen
c               vrtlst
c
c       PARAMETERS:
c               nparts-         the number of parts
c               part-           the list of parts
c               bbox-           an array specifying the bounding boxes of the
c                               parts
c               ptlist-         the ordered parts list

c       VARIABLES:
c               coord,fcoord- vertex coordinates
c               rmin,rmax- the minimum,maximum coordinates of the bounding box
c               scoord,scord2- screen coordinates
c               vlist-  list of adjacent vertices
c               best-   .true. when no more vertices are needed to define the
c                       bounding box
c
c
c******************************************************************************
c


        integer part(*)
        integer ptlist(2,*)
        integer vlist(20)
        integer rmin(3),rmax(3)
        integer bbox(6,*)
        integer*2 scoord(3),scord2(3)
        real  coord(3),fcoord(3)
        integer iftemp(2)
        logical best,istat


c       calculate the bounding boxes for all the parts

        do 10 ipart=1,nparts
           call fplist(part(ipart),iface,iend)
           call vrtlst(iface,istart,iend)
           call getvrt(istart,ifirst)
           call getcrd(ifirst,fcoord)
           call screen(1,fcoord,scoord,istat)

c          Test extents along each axis.

           do 20 iaxis=1,3
              rmin(iaxis)=scoord(iaxis)
              rmax(iaxis)=scoord(iaxis)
              do 21 indl=1,2
                 best=.false.
                 ivert=ifirst

c                Test all the adjacent vertices. If none increase the maximum
c                extents along the axis,
c                then no more vertices need be checked due to convexity.

c             WHILE (notbest)
100              if (best) goto 200
                    best=.true.
                    call ajvrts(ivert,vlist,nverts)
                    do 25 index=1,nverts
                       call getcrd(vlist(index),coord)
                       call screen(1,coord,scord2,istat)
                       if (indl.eq.1) then
                          if (scord2(iaxis).lt.rmin(iaxis)) then
                             rmin(iaxis)=scord2(iaxis)
                             ivert=vlist(index)
                             best=.false.
                          endif
                       else
                          if (scord2(iaxis).gt.rmax(iaxis)) then
                             rmax(iaxis)=scord2(iaxis)
                             ivert=vlist(index)
                             best=.false.
                          endif
                       endif
25                  continue
                    goto 100
200              continue
c             ENDWHILE

21            continue
20         continue


           index=part(ipart)

           bbox(1,index)=rmin(1)
           bbox(2,index)=rmax(1)
           bbox(3,index)=rmin(2)
           bbox(4,index)=rmax(2)
           bbox(5,index)=rmin(3)
           bbox(6,index)=rmax(3)

           ptlist(1,ipart)=index
```

```
10      continue


c       sort the parts by depth

        do 50 i=1,nparts-1
           do 55 j=i+1,nparts
              if (bbox(6,ptlist(1,j)).gt.bbox(6,ptlist(1,i))) then
                 do 60 k=1,2
                    iftemp(k)=ptlist(k,j)
                    ptlist(k,j)=ptlist(k,i)
                    ptlist(k,i)=iftemp(k)
60               continue
              endif
55         continue
50      continue


        return
        end
        subroutine osort2(pindex,qindex,ptlist,iflag)

c       Find the first polyhedral object in the object list which obscures the
c       object at the head of the list.

c       CALLED BY-
C               ohsurf

c       SUBROUTINES CALLED-
c               none

c       PARAMETERS-
c               pindex- index of face P
c               qindex- the index of the last obscuring object
c               ptlist- list of objects including status
c               iflag-  index of first obscuring object
c*********************************************************************************

        integer pindex,qindex,iflag
        integer ptlist(2,*)
        integer reset
        logical stop

        reset=0
        iflag=0


        icount=0
        stop=.false.
        do 10 index=pindex+1,qindex
           if (.not.stop) then
              if (ptlist(2,index).ne.reset) then
                 if (iflag.eq.0) then
                    iflag=index
                    stop=.true.
                 endif
              endif
           endif
10      continue

        return
        end
```

```
      subroutine obswap(pindex,nparts,iflag,ptlist,bbox)

c     This subroutine inserts the current object P into the object list
c     according
c     to its maximum screen depth, and replaces it with the object whose index
c     is 'iflag'.

c     CALLED FROM-
c             ohsurf

c     SUBROUTINES CALLED-
c             none

c     PARAMETERS-
c             pindex- index of the current head of the object list, which
c                     contains the current object P.
c             nparts- the number of objects in the list
c             iflag-  the index of the object which will be moved to the
c                     head of the object list.
c             ptlist- the list of objects
c             bbox-   the object bounding boxes

c*****************************************************************************


      integer pindex, temp(2)
      integer ptlist(2,*)
      integer bbox(6,*)
      logical insert

c     Save the current face P and replace it in the face list by face Q.

      if (pindex.ne.iflag) then
      do 10 i=1,2
         temp(i)=ptlist(i,pindex)
         ptlist(i,pindex)=ptlist(i,iflag)
10    continue

c     Remove face Q from its previous location in the face list.

      do 20 index=iflag,nparts-1
         do 30 i=1,2
            ptlist(i,index)=ptlist(i,index+1)
30       continue
20    continue

c     Insert face P in the correct place in the face list according to
c     maximum screen depth.

      insert=.false.
      do 40 icount=pindex+1,nparts-1
         if (.not.insert) then
            if(bbox(6,temp(1)).ge.bbox(6,ptlist(1,icount))) then
               do 41 i=nparts-1,icount,-1
                  do 42 j=1,2
                     ptlist(j,i+1)=ptlist(j,i)
```

```
42                   continue
41                continue
                  do 45 i=1,2
                     ptlist(i,icount)=temp(i)
45                continue
                  insert=.true.
               endif
            endif
40       continue

c        If face P hasn't been inserted yet, then it goes at the end of the list.

         if (.not.insert) then
            do 50 i=1,2
               ptlist(i,nparts)=temp(i)
50          continue
         endif
         endif

         return
         end
```

```
      subroutine ohsurf(nparts,ptlist,bbox,oparts)

c     Hidden surface removal using the Newell/Newell/Sancha algorithm
c     modified to sort convex polyhedra. Unlike the case for general face
c     sorting, it is not possible for two disjoint polyhedra to obscure
c     each other. Intersecting polyhedra are not handled.

c     CALLED FROM: psort

c     SUBROUTINES CALLED:
c             initgp
c             ohtest
c             osort2
c             obswap

c     PARAMETERS:
c             nparts- the number of objects to be sorted
c             ptlist- the list of objects and their status
c             bbox-   the bounding boxes of the objects
c             oparts- the resulting priority ordered list of objects

c     VARIABLES:
c             pindex-         the index of the current head of the object li
c                             This is also the number of parts currently
c                             sorted
c             qindex-         the upper bound index defining the set of
c                             objects to be tested
c             iflag-          the id of the first part which obscures
c                             the object at the head of the list
c
c*****************************************************************************
c

      integer oparts(*)
      integer ptlist(2,*)
      integer bbox(6,*)
      logical stop,result
      integer pindex,qindex
      integer reset

c     initialize parameters

      pindex=1

c     initialize constants

      reset=0

c     initialize global data base parameters

      call initgp

c     repeat until all objects have been ordered
```

```
c       WHILE (pindex.lt.nparts)

        ptlist(2,pindex)=reset
10      if (pindex.ge.nparts) goto 20

c       perform a hierarchy of tests on the current object against all of the
c       other objects

           call ohtest(pindex,qindex,nparts,ptlist,bbox,stop)


           if(.not.stop) then


c       Get the first object which is obscured by the current object.

              call osort2(pindex,qindex,ptlist,iflag)

              if (iflag.le.0.or.iflag.ge.100) then
                 write(*,*) 'ERROR IN OHSURF- iflag error,iflag=',iflag
              endif

c       swap the current object with an obscured object.

              call obswap(pindex,nparts,iflag,ptlist,bbox)

c       insert the current object into the ordered object list and remove it
c       from the list of objects to be sorted

           else
              oparts(pindex)=ptlist(1,pindex)
              ptlist(2,pindex)=reset
              pindex=pindex+1
           endif

        if (pindex.ge.100) then
           write(*,*) 'ERROR IN OHSURF- pindex too large'
        endif

c       END WHILE

        goto 10
20      continue

c       the last remaining object goes at the end of the list

        oparts(pindex)=ptlist(1,pindex)

        return
        end
```

```
      subroutine ohtest(pindex,qindex,nparts,ptlist,bbox,stop)

c     Perform a hierarchy of hidden surface tests on a candidate object P
c     against a set of objects {Q} according to a modified
c     Newell/Newell/Sancha algorithm. The following tests are performed:

c             1. Overlap in depth (z)
c             2. Overlap in x,y
c             3. Order of objects relative to their separating plane
c             4. Overlap of their projections on the screen


c     CALLED FROM:
c             ohsurf

c     SUBROUTINES CALLED:
c             disjnt
c             splsid

c     PARAMETERS:
c             pindex- the index of the current candidate face P in the list
c             qindex- the index of the last face of the set {q} in the list
c             nparts- the number of parts to be tested
c             ptlist- the ordered parts list
c             bbox-   an array specifying bounding boxes for the parts
c             stop-   .true. if all of the tests succeed and face P can be
c                     output to the ordered face list

c     Variables:
c             fpair-  a pair of objects to be compared
c             obscur- .true. if the object 'Q' is possibly obscured by the
c                     candidate object 'P'
c             reset-  .true. if 'Q' is not obscured by 'P'
c
c*****************************************************************************

      integer fpair(2)
      integer ptlist(2,*)
      integer pindex,qindex
      integer reset,obscur
      integer bbox(6,*)
      integer pxmin,pxmax,pymin,pymax,pzmin,pzmax
      logical stop,result

c     initialize parameters

      reset=0
      obscur=1

      qindex=pindex
      ptlist(2,pindex)=reset


c     find the set of faces {Q} which overlap P in z

c     WHILE (bbox(5,ptlist(1,qindex)).lt.bbox(6,ptlist(1,pindex)))
```

```
        pzmin=bbox(5,ptlist(1,pindex))
        stop=.true.
20      if (bbox(6,ptlist(1,qindex+1)).lt.pzmin.or.qindex+1.gt.nparts) goto 10
           qindex=qindex+1
           ptlist(2,qindex)=obscur
           stop=.false.
c       END WHILE
        goto 20
10      continue

c       test bounding boxes of P and Q in x and y.

        if (.not.stop) then
        stop=.true.
        pxmax=bbox(2,ptlist(1,pindex))
        pxmin=bbox(1,ptlist(1,pindex))
        pymin=bbox(3,ptlist(1,pindex))
        pymax=bbox(4,ptlist(1,pindex))
        do 30 index=pindex+1,qindex
           if (ptlist(2,index).eq.obscur) then
              if (bbox(1,ptlist(1,index)).gt.pxmax.or.
     *            bbox(2,ptlist(1,index)).lt.pxmin.or.
     *            bbox(3,ptlist(1,index)).gt.pymax.or.
     *            bbox(4,ptlist(1,index)).lt.pymin) then
              ptlist(2,index)=reset

              else
                 stop=.false.
              endif
           endif
30      continue
        endif


c       test if P is wholly on far side of Q and if Q is wholly on near
c       side of P.


        if (.not.stop) then
           stop=.true.
           fpair(1)=ptlist(1,pindex)
           do 50 index=pindex+1,qindex
              if (ptlist(2,index).eq.obscur) then
                 fpair(2)=ptlist(1,index)
                 call splsid(fpair,result)
                 if (result) then
                    ptlist(2,index)=reset
                 else
                    stop=.false.
                 endif
              endif
50         continue
        endif
```

```
c       test if the projections of P and Q are disjoint

        if (.not.stop) then
           stop=.true.
           fpair(1)=ptlist(1,pindex)
           result=.false.
           do 60 index=pindex+1,qindex
              if (ptlist(2,index).ne.reset) then
                 fpair(2)=ptlist(1,index)
                 call disjnt(fpair,result)
                 if (result) then
                    ptlist(2,index)=reset
                 else
                    stop=.false.
                 endif
              endif
60         continue
        endif

        return
        end
```

```
      subroutine pface(face,vert,vface,dist)

c     Project a vertex onto a face plane.

c     Called from: finbes

c     SUBROUTINE CALLS:
c           facvrt
c           getcrd
c           getnml

c     PARAMETERS:
c           face-   the ID of the test face
c           vert-   the ID of the vertex to be projected
c           vface-  the coordinates of the vertex projected onto the face
c                   plane
c           dist-   the normal distance of the vertex from the face plane
c****************************************************************************

      integer face,vert
      real vface(3),dist
      real fcoord(3),normal(3),vcoord(3)


c     Get the coords of a point on the face, the normal coordinates,
c     and the coords of the test vertex.

      call facvrt(face,fcoord)
      call getnml(face,normal)
      call getcrd(vert,vcoord)


c     Find the normal distance from the vertex to the face plane

      dist=0.
      do 10 i=1,3
         dist=dist+(vcoord(i)-fcoord(i))*normal(i)
10    continue


c     Find the projected vertex coordinates.

      do 20 i=1,3
         vface(i)=vcoord(i)-dist*normal(i)
20    continue

      return
      end
      subroutine picksm(nsave,nobj,object,buffer,npick,istat)

c     Pick a seam.

c     Parameters:        nsave-  the number of objects to be saved.
c                        nobj-   number of objects defined.
c                        object- array of object ids.
c                        buffer- array of picked objects.
c                        npick-  number of objects picked.
c                        istat-  .true. if a pick has been initiated.

c     Subroutines called: none

c     Graphics calls: callob,endpic,multma,pick,polarv,window

c     Called from: mkseam,smpick

c     Common blocks modified: reset

c**********************************************************************************************

$include /usr/include/fgl.h
$include /usr/include/fdevice.h

      real winsiz(6)
      integer*2 azim,inc,twist
      real vdist
      real ftran1(4,4),ftran2(4,4)
      integer object(*),buffer(*)
      logical istat
      logical reset

      integer*2 ibut,data


      common /view/vdist,azim,inc,twist
      common /winsiz/winsiz
      common /ftrans/ftran1,ftran2
      common /reset/reset


      ibut=qread(data)
      istat=.false.
      if (data.ne.0) then
         if (ibut.eq.mouse1) then
            istat=.true.
            call pick(nsave)
            call window(winsiz(1),winsiz(2),winsiz(3),winsiz(4),winsiz(5),
     *                     winsiz(6))
            call polarv(vdist,azim,inc,twist)
            call multma(ftran2)
            do 10 i=1,nobj
               call callob(object(i))
10          continue
            npick=endpic(buffer)
         endif
      endif

      return
      end
```

```
        subroutine plist(type,parts,nparts,colist,icolst,dmin)

c       Find the separating planes for a set of parts and generate a separating
c       plane list.

c       Called from: drpath,drwkst,intest

c       SUBROUTINES CALLED:
c               facvrt
c               finbes
c               finsep
c               getnml
c               getsep

c       GRAPHICS CALLS:
c               charst,cmov2,color,draw,move,ortho,popmat,pushma,window

c       PARAMETERS:
c               type-   0- stop when the first separating plane is found.
c                          This is sufficient for hidden surface removal.
c                       1- find the best separating plane.  This allows the
c                          minimum distance to be found.
c               parts-  the list of parts to be processed.
c               nparts- the number of parts
c               colist- a list of part pairs which intersect.
c               icolst- the number of intersecting pairs.
c               dmin-   the minimum distance between the closest parts

c       Variables:
c               bestf- best separating face between two parts
c               bestv- closest vertex of second part to best separating face
c                       of first part
c               dist-   minimum distance between two parts
c               istart- starting point in 'csplan'
c               nplane- separating plane number
c               pair-   the pair of parts currently being tested
c               status- 0- 'bestf' is a separating plane
c                       1- no faces are separating planes
c               ncsep-  number of known separating planes

c

c***********************************************************************************

$include seplane.com

        integer type, parts(*)
        integer colist(*)
        real coord(3),normal(3)
        integer pair(2),bestf,bestv,status
        real temp(3)
        logical pstat,stop
        real vdist
        integer*2 azim,inc,twist
```

```
      common /view/vdist,azim,inc,twist

      dmin=1000.

c     Test every pair of parts. The parts are ordered by increasing id.

      istart=1
      icolst=0
      do 20 ind1=1,nparts
         call getsep(parts(ind1),nstart,imax)
         pair(1)=parts(ind1)
         do 30 ind2=1,ind1-1
            nplane=nstart+parts(ind2)-1
            pair(2)=parts(ind2)

c     Check if the two parts have an invariant, known separating plane.
c     The separating planes are ordered, so we start the search from
c     the last separating plane found.

            stop=.false.
            do 11 iplan=istart,ncsep
               if (.not.stop) then
                  if (csplan(iplan).eq.nplane) then
                     istart=iplan
                     stop=.true.
                   endif
               endif
11          continue

c     Otherwise, find a separating plane.

            if (.not.stop) then
            if (type.eq.0) then

c              Find any separating plane.

               call finsep(0,pair,index,bestf,bestv,status,pstat)
               if (status.eq.0) then

c                 A face is a separating plane.

                  call facvrt(bestf,coord)
                  call getnml(bestf,normal)
                  dist=10000.
               else

c                 Find a separating plane which is not a face.

                  call finbes(pstat,bestf,bestv,coord,normal,dist)
               endif

            else

c              Find the best separating plane, and the exact separating
c              distance and connecting vector.
```

```
              call finsep(1,pair,index,bestf,bestv,status,pstat)
              call finbes(pstat,bestf,bestv,coord,normal,dist)
            endif

c           Record the parts which interfere.

            if (dist.lt.0.) then
              do 50 i=1,2
                icolst=icolst+1
                colist(icolst)=pair(i)
50            continue
            endif

c       Store the vectors and normals defining the separating planes.

            do 10 i=1,3
              vlist(i,nplane)=coord(i)
              if (index.eq.1) nlist(i,nplane)=normal(i)
              if (index.eq.2) nlist(i,nplane)=-normal(i)
10          continue


c       For debugging, draw the shortest separating vector on the screen

            if (type.eq.1) then
              call pushma
              call polarv(vdist,azim,inc,twist)
              call pushat
              call color(i2(13))
              call move(coord(1),coord(2),coord(3))
              do 100 i=1,3
                temp(i)=coord(i)+normal(i)
100           continue
              call draw(temp(1),temp(2),temp(3))
              call gflush
              call popatt
              call popmat
            endif

c           Update the minimum separating distance found so far.

            if (dist.lt.dmin) dmin=dist

            endif

30        continue
20      continue

        return
        end
```

```
         subroutine posobs(lunit,idweld,posit,nmax,nobjs)

c       Generate a list of inter-weld positions by reading a file.

c       Parameters:     lunit- an available logical unit.
c                       idweld- the id of the weld following the path.
c                       posit-  array of pointers into the positions table.
c                       nmax-   the dimension of 'posit'
c                       nobjs-  the number of graphical objects.

c       Called from: dfpath

c       Subroutines called: none

c       Common blocks modified: none

c       Files read: itable,ilocat

c       Files written: none

c*******************************************************************************

$include /usr/include/fgl.h

        character*20 itable
        integer idweld,nobjs
        real values(10,3)
        integer posit(*)
        integer recl(20)

        common /itable/itable
        common /reclen/recl

c       Open the pointer file and read the starting and ending records
c       for the inter-weld positions.

        open(unit=lunit,file=itable,form='unformatted',access='direct',
     *       status='old',recl=recl(7))
        read(lunit,rec=idweld) nstart,nend
        close(lunit)

        nobjs=0

c       Write the record numbers into an array.

        do 10 ind=nstart,nend

           nobjs=nobjs+1

           if (nobjs.le.nmax) then
              posit(nobjs)=ind
           else
              write(*,*) 'MAXIMUM NUMBER OF OBJECTS EXCEEDED IN posobs'
           endif
```

```
10      continue

        return
        end
        SUBROUTINE PROCES
C
C       THIS PROGRAM READS THE WELDING PARAMETERS, R, N1, N2 FROM CAD
C       FILE, CALCULATES ARCLENGTH S, THE POSITION VECTOR R, AND THE
C       seam FRAME AXES T, A, B.
C       THE RESULTS ARE STORED IN AN OUTPUT FILE.
C
C

c       Subroutines called: arclen,norm,nxprod,tangen

c       Called from: main

c       Common blocks modified: none

c       Files read: cadlst,cad

c       Files written: procad


C
C***************************************************************************
C
        CHARACTER*20 CAD
        character*20 cadlst,procad
        REAL R(3,3),N1(3),N2(3),T(3),A(3),B(3),S(3)
        integer recl(20)

        COMMON /CAD/CAD
        COMMON /CADLST/CADLST
        common /procad/procad
        common /nseams/nseams
        common /reclen/recl

        DATA R/9*0./
C



        open(unit=1,file=cadlst,form='unformatted',access='direct',
     *          status='old',recl=recl(2))
        open(unit=2,file=cad,form='unformatted',access='direct',
     *          status='old',recl=recl(1))
        open(unit=3,file=procad,form='unformatted',access='direct',
     *          status='new',recl=recl(4))

        do 15 ind=1,nseams
C
C*****  INITIALIZE ARCLENGTH 'S', FLAG 'IFLAG'
C
        S(3)=0.
        IFLAG=0
C
C
        READ(1,rec=ind) nstart,NPOINT

        DO 10 IND1=nstart,nstart+npoint
```

```
C
C
       IF (IND1.EQ.(nstart+NPOINT)) THEN
C*****     LAST POINT
           IFLAG=3
       ELSE

           read(2,rec=ind1) (r(3,i),i=1,3),(n1(j),j=1,3),(n2(k),k=1,3)
C
C*****     READ FIRST TWO POINTS BEFORE BEGINNING CALCULATIONS.
C*****     SET FLAGS IF SECOND POINT OR LAST POINT IS READ.
C*****     THESE POINTS REQUIRE A MODIFIED PROCEDURE.
C
C*****     FIRST POINT
C
           IF(IND1.EQ.nstart) then
              iflag=2
C
C*****     SECOND POINT
C
           elseIF(IND1.EQ.nstart+1) then
              iflag=1
           endif
       ENDIF
C
       if (iflag.ne.2) then
C
C*****     CALCULATE ARCLENGTH S AT POSITION R. 'DELS' IS THE CHANGE IN
C*****     ARCLENGTH FROM R(I-1) TO R(I)
C
           CALL ARCLEN(R,DELS)
           S(3)=S(2)+DELS
C
C*****     CALCULATE THE TANGENT VECTOR 'T' AT POSITION 'R'
C
           CALL TANGEN(R,S,T,IFLAG)
C
C*****     CALCULATE THE NORMAL VECTOR 'A' FROM THE SURFACE NORMALS
C*****     'N1','N2'
C
           CALL NORM(N1,N2,A)
C
C
C*****     CALCULATE THE LATERAL VECTOR 'B', B=A X T (CROSS PRODUCT)
C
           CALL NXPROD(A,T,B)
C
C*****     A=T X A (CROSS PRODUCT). THIS STEP ENSURES AN ORTHOGONAL
C*****     FRAME.
C
           CALL NXPROD(T,B,A)
C
```

```
C*****        WRITE RESULTS INTO OUTPUT FILE.
C
              write(3,rec=indl-1) s(2),(r(2,i),i=1,3),(t(j),j=1,3),(a(k),k=1,3),
     *                                 (b(l),l=1,3)
C
C*****        UPDATE CURRENT POSITION, I, BY REDEFINING POINT I AS
C*****        POINT I-1.
C
              DO 300 K=1,3
                 R(1,K)=R(2,K)
300           CONTINUE
              S(1)=S(2)
C
           endif
C
C*****     SIMILARLY, REDEFINE POINT I+1 AS POINT I.
C
           DO 301 K=1,3
              R(2,K)=R(3,K)
301        CONTINUE
           S(2)=S(3)
C
10      CONTINUE
15      continue
C
C
C*****  CLOSE INPUT AND OUTPUT FILES.
C
        close(1)
        close(2)
        close(3)
C
        RETURN
        END
```

```
      subroutine psort(part,number,oparts)

c     Order a set of disjoint polyhedral parts from back to front so that
c     hidden surfaces are removed.

c     CALLED FROM:
c             drwkst
c
c     SUBROUTINES CALLED:
c             ohsurf
c             obsort

c     PARAMETERS:

C             number- the number of parts to be shown
c             part-   a table of part ID's.
c             opart-  the ordered parts list

c     Variables:

c             ptlist- parts list sorted by depth
c             bbox-   the bounding boxes of the parts
c***********************************************************************

      integer part(*)
      integer oparts(*)
      integer ptlist(2,100)
      integer bbox(6,100)


c     Sort the parts by depth

      call obsort(number,part,bbox,ptlist)

c     Hidden surface removal algorithm- generate an ordered parts list

      call ohsurf(number,ptlist,bbox,oparts)

      return
      end
      SUBROUTINE RDMODL
C
C     THIS subroutine READS A POLYHEDRAL MODEL FROM A FILE AND
C     INITIALIZES THE DATA STRUCTURES in common blocks 'common' and 'seplane'.
c     The files are formatted and sequential for easy editting and debugging.
c     The files must match the 'read' statements in this subroutine record
c     for record.

c     Called from: main

c     Subroutines called: none

c     Files read: geom,setup

c*******************************************************************************

$include common.com
$include seplane.com

C
      character*20 GEOM
      character*20 setup
      integer part(100),wkpiec(100),robotg(100)
      integer number,wnumb,rnumb

      common /parts/part,number
      common /wkpiec/wkpiec,wnumb
      common /robotg/robotg,rnumb
c
      common /GEOM/geom
      common /SETUP/setup

C
C     Open THE GEOMETRY FILE
C
      open(unit=1,file=geom,status='old')
C
      read(1,*)
      read(1,*)
      read(1,*)

c     Read the number of clusters.

      read(1,*) nclust

c     First part, no.parts, first face, no.faces, first vert., no.verts.

      do 5 i=1,nclust
         read(1,*) index,(clustr(j,i),j=1,6)
5     continue

      read(1,*)
      read(1,*)
      read(1,*)
```

```
c       Number of parts.

        READ(1,*) NPT

c       First face, no. of faces.

        DO 10 I=1,NPT
           READ(1,*) index,(PARTS(J,I),J=1,2)
10      CONTINUE

        read(1,*)
        read(1,*)
        read(1,*)

c       Number of known separating planes.

        read(1,*) ncsep

c       Separating plane index.

        do 15 i=1,ncsep
           read(1,*) itemp,itemp,itemp,csplan(i)
15      continue
        csplan(ncsep+1)=1000

c       Fill separating plane pointer table.

        icount=1
        ptabl(2)=1
        do 7 i=1,98
           icount=icount+i
           ptabl(i+2)=icount
7       continue

        read(1,*)
        read(1,*)

c       Read separating plane coords and normals.

        do 16 i=1,ncsep
           read(1,*) ind,(fvlist(j,i),j=1,3),(fnlist(j,i),j=1,3)
16      continue

C
c       Read total number of faces.

        read(1,*)
        READ(1,*) NFACE
C
        read(1,*)
        read(1,*)
        read(1,*)

c       Read total number of vertices.

        READ(1,*) NVL
```

```
      icount=1
c
c     Read vertices of each face into a 1-d array and store pointers
c     in an array.
c
      DO 30 I=1,nface
         READ(1,*) index,number,(VERTS(j),j=icount,icount+number-1)
         faces(1,i)=icount
         faces(2,i)=number
         icount=icount+number
30    CONTINUE
C
      read(1,*)
      read(1,*)
      read(1,*)
c
c     Read number of unique vertices.
c
      READ(1,*) NVERT
c
c     Read vertex coordinates.
c
      DO 40 I=1,NVERT
         READ(1,*) index,(VCOORD(J,I),J=1,3)
C  VCOORD(4,I)=1.
40    CONTINUE
c
c
c     Read faces adjacent to vertices.
c
      read(1,*)
      read(1,*)
      read(1,*)
      READ(1,*) NFACE2
      icount=1
      DO 50 I=1,nvert
         READ(1,*) index,number,(FACES2(j),j=icount,icount+number-1)
         overts(1,i)=icount
         overts(2,i)=number
         icount=icount+number
50    CONTINUE
c
c
c     Read faces adjacent to faces.
c
      read(1,*)
      read(1,*)
      read(1,*)
      read(1,*) nface3
      icount=1
      do 55 i=1,nface
         read(1,*) index,number,(faces3(j),j=icount,icount+number-1)
         faces(3,i)=icount
         faces(4,i)=number
         icount=icount+number
55    continue
```

```
C
c     Read vertices adjacent to vertices.

      read(1,*)
      read(1,*)
      read(1,*)
      READ(1,*) NVADJ
      icount=1
      DO 60 I=1,nvert
        READ(1,*) index,number,(ajvert(j),j=icount,icount+number-1)
        overts(3,i)=icount
        overts(4,i)=number
        icount=icount+number
60    CONTINUE
C
c     Read coordinates of face normals.

      read(1,*)
      read(1,*)
      read(1,*)
      READ(1,*) NFNORM
      DO 70 I=1,NFNORM
        READ(1,*) index,(FNORMS(J,I),J=1,3)
C  FNORMS(4,I)=1.
70    CONTINUE

      close(1)


c     Open the setup file.

      open(unit=1,file=setup,status='old')

c     Read the parts list for the workstation, to be tested for interference.

      read(1,*)
      read(1,*) number
      read(1,*) (part(i),i=1,number)

c     Read the list of parts to be shown.

      read(1,*)
      read(1,*) wnumb
      read(1,*) (wkpiec(i),i=1,wnumb)

c     Read the list of parts to be highlighted during program replay.

      read(1,*)
      read(1,*) rnumb
      read(1,*) (robotg(i),i=1,rnumb)

C
      CLOSE(1)
C
```

```
      RETURN
      END
```

```
        subroutine rempos(lunit,idweld,index)

c       Remove a position from an inter-weld path.
c       The position data is removed from the file 'ilocat', and the
c       positions pointer file 'itable' is revised.

c       Parameters:     lunit-  logical unit number
c                       idweld- the id of the weld associated with the path
c                       index-  the starting position of the path to be removed

c       Subroutines called: none

c       Called from: editpa

c       Files 'itable', 'ilocat' are read and modified.

c*****************************************************************************************

        character*20 itable,ilocat
        integer nrdof(3),savend
        real save1(10,3),save2(10,3)
        integer recl(20)


        common /itable/itable
        common /ilocat/ilocat
        common /nrdof/nrdof
        common /reclen/recl


c       Update the position pointers table.

        open(unit=lunit,file=itable,form='unformatted',access='direct',
     *        status='old',recl=recl(7))

        read(lunit,rec=idweld) nstart,nend
        savend=nend
        nend=nend-1
        write(lunit,rec=idweld) nstart,nend

        icount=idweld

10      continue
        icount=icount+1
        read(lunit,rec=icount) nstart,nend
        if (nstart.ne.0) then
           savend=nend
           nstart=nstart-1
           nend=nend-1
           write(lunit,rec=icount) nstart,nend
           goto 10
        endif

        write(lunit,rec=icount) 0,0

        close(lunit)
```

```
c       Remove a position from the positions table.

        open(unit=lunit,file=ilocat,form='unformatted',access='direct',
     *      status='old',recl=recl(8))

        do 20 ind=index+1,savend

           read(lunit,rec=ind) (savel(i,1),i=1,nrdof(1)),
     *     (savel(i,2),i=1,nrdof(2)),time

           write(lunit,rec=ind-1) (savel(i,1),i=1,nrdof(1)),
     *        (savel(i,2),i=1,nrdof(2)),time

20      continue

        close(lunit)

        return
        end
        subroutine revdir(idseam,sequen,seams,grobjs)

c       Reverse the direction of a previously defined seam.
c       The common flag 'update' is set.

c       Parameters:     idseam- the id of the picked seam
c                       sequen- segments pointer table for the seam
c                       seams-  the seam segments
c                       grobjs- table of graphical objects for the seams

c       Subroutines called: delobj,drsequ

c       Called from: smpick

c       Parameters modified: seams,grobjs

c       Common blocks modified: update
c
c       Files read: none

c******************************************************************************

        integer idseam,sequen(2,*),seams(3,*),grobjs(*)
        integer files(3,10)
        logical update

        common /update/update


c       Find the start and number of seam segments.

        istart=sequen(1,idseam)
        nfiles=sequen(2,idseam)


c       Save the segments temporarily.

        iend=istart+nfiles-1

        index=0
        do 10 ind=iend,istart,-1

           index=index+1
           files(1,index)=seams(1,ind)
           files(2,index)=seams(3,ind)
           files(3,index)=seams(2,ind)

10      continue


c       Copy the segments in reverse order.

        index=0
        do 20 ind=istart,iend
```

```
           index=index+1
           do 30 i=1,3

              seams(i,ind)=files(i,index)
30         continue
20      continue

c       Update the graphical object, showing the new direction.

        call delobj(grobjs(idseam))
        call drsequ(files,nfiles,iobj)
        grobjs(idseam)=iobj

        update=.true.

        return
        end
```

```
      subroutine rmseam(idseam,grobjs)
c     Remove a seam segment from the user defined set.
c     Parameters:     idseam- the seam id
c                     grobjs- a table of graphical objects
c     Subroutines called: none
c     Called from: smpick
c     Parameters modified: grobjs
c     Common blocks modified: update
c*************************************************************************
      integer idseam,grobjs(*)
      logical update
      common /update/update
      update=.true.
      call delobj(grobjs(idseam))
      grobjs(idseam)=0
      return
      end
      subroutine screen(n,pretra,postra,istat)

c     This subroutine emulates the IRIS command 'screen' which doesn't work.
c     Screen transforms an array of world coordinate vectors into screen
c     coordinates including perspective depth into the screen.
c
c     CALLED FROM: cvhull,obsort

c     SUBROUTINES CALLED:     none

c     Graphics calls: transf

c     PARAMETERS:
c             n-      dimension of arrays
c             pretra- array of real world coordinate vectors
c             postra- array of vectors transformed into integer screen
c                     coordinates
c             istat-  a dummy logical array to match the IRIS command.
c*****************************************************************************


      integer n
      integer*2 postra(3,n)
      real pretra(3,n)
      logical istat(n)
      real winsiz(6)
      integer*2 left,right,bottom,top

      real templ(4),temp(4)

      common /winsiz/winsiz
      common /viewpo/left,right,bottom,top
      common /screen/diff1,diff2,diff3,sum1,sum2,sum3,vcx,vsx,vcy,vsy,vcz,vsz

      do 5 i=1,3
         temp(i)=pretra(i,n)
5     continue
      temp(4)=1.

      call transf(1,temp,templ)

      do 10 i=1,n
         postra(1,n)=int(templ(1)/templ(4)*vsx+vcx)
         postra(2,n)=int(templ(2)/templ(4)*vsy+vcy)
         postra(3,n)=int(templ(3)/templ(4)*vsz+vcz)
10    continue


      return
      end
      subroutine setmen

c     Set up a menu area on the screen.

      integer*2 black,white

      black=0
      white=11

      call ortho2(750.,1023.,0.,767.)
      call viewpo(i2(750),i2(1023),i2(0),i2(767))

      call color(black)
      call rectfi(750,0,1023,767)
      call color(white)
      call recti(751,0,1023,767)

      return
      end
```

```
        subroutine smpick(ipick,icount,nseq,sequen,seams,grobjs)

c       Interactive seam picking and automatic weld path generation.
C
c       Parameters:     ipick-  list of previously welded seams.
c                       icount- number of welded seams.
c                       nseq-   number of segments in a seam definition
c                       sequen- array of segment pointers
c                       seams-  segment ids, and pointers into reference seam
c                               file
c                       grobjs- array of seam graphical object ids.

c       Subroutines called: drtext,genpat,picksm,revdir,rmseam,unweld

c       Graphics calls: pushma,popmat,finish,ringbe

c       Called from: main

c       Parameters modified: none

c       Common blocks modified: flags1,flags3,rmweld,update,mkpath,idseam,
c                               idweld,path,makep,start2
c       Variables:
c               defwel- .true. to generate a weld trajectory for a picked seam
c               makep-  .true. to activate inter-weld path mode
c               mkpath- .true. to pick inter-weld path
c               mvtabl- .true. for synchronous positioner motion
c               pickfl- .true. if the picking mode is activated
c               pstat-  .true. if a graphical object was successfully picked
c               remove- .true. to remove a previously defined seam
c               revdir- .true. to reverse the direction of a seam
c               rmweld- .true. to unweld a welded seam
c               idseam- the seam id
c               idweld- the weld id
c               start2- .true. at start of a joint interpolated move between
c                       two inter-weld positions
c               update- .true. to update seam and weld data
c
c
c**************************************************************************

$include /usr/include/fdevice.h
$include /usr/include/fgl.h

        INTEGER IPICK(*)
        integer sequen(2,*),seams(3,*),grobjs(*),nseq

        integer buffer(300)
        integer idseam
        integer files(3,10),nfiles
C

        character*10 string


        logical replay,shdist
        logical pickfl,tinit,tfinal
        logical solid,shrobt,keyon
        logical pstat
        logical remove,direv
        logical rmweld
        logical update
        logical mkpath
        logical makep
        logical start2
        logical mvtabl
        logical defwel
c
        common /flags1/keyon,replay,shdist,pickfl,solid,shrobt
        common /nsave/nsave
        common /flags3/remove,direv
        common /rmweld/rmweld
        common /update/update
        common /mkpath/mkpath
        common /idseam/idseam
        common /idweld/idweld
        common /makep/makep
        common /start2/start2
        common /flags2/mvtabl,tinit,tfinal
        common /defwel/defwel
C
C
C
C*****  PICK THE NEXT SEAM TO BE WELDED
C
C
c       Enter picking mode

        if (pickfl) then
           call drtext('PICK SEAM- PRESS MOUSE1 ',' ')

c       Pick seam with mouse

           call pushma
           call picksm(nsave,nseq,grobjs,buffer,npick,pstat)
           call popmat


c       Identify the picked seam. For a successful pick, the returned object
c       must be in the set of user-defined seam objects.

           if (pstat) then

              pickfl=.false.
              if (npick.gt.1) then
                 iobj=buffer(4)
                 do 10 i=1,nseq
                    if (grobjs(i).eq.iobj) then
                       idseam=i
                    endif
10               continue
```

```
              else
                 idseam=-1
              endif
              call finish

C
C**  IF (PICK UNSUCCESSFUL)
C
              if (idseam.eq.-1) then
C
                 call ringbe
                 call drtext('PICK UNSUCCESSFUL-TRY AGAIN  ',' ')
                 remove=.false.
                 direv=.false.
                 rmweld=.false.

              else

C
C*****  CHECK IF THE SEAM HAS ALREADY BEEN PICKED.
C
                 if (.not.rmweld) then
C
C*****     'IPICK' IS AN ARRAY CONTAINING ALL PICKED SEAMS.
C*****     'ICOUNT' IS THE NUMBER OF SEAMS ALREADY PICKED.
C*****     'IDSEAM' IS THE CURRENT SEAM ID
C*****     'IPREV' IS A STATUS FLAG. IF 'IDSEAM' ALREADY PICKED, IPREV=1

                    CALL CHECK(IPICK,ICOUNT,IDSEAM,IPREV)

                 else
                    iprev=0
                 endif

C
C**  IF (SEAM PREVIOUSLY PICKED)

                 if (.not.mkpath.and.iprev.ne.0) then
C
                    call ringbe
                    call drtext('SEAM ALREADY PICKED- TRY AGAIN  ',' ')
c
                 elseif (mkpath.and.iprev.eq.0) then

                    call ringbe
                    call drtext('THAT SEAM HASN''T BEEN WELDED- TRY AGAIN',' ')

                 else

C
C**  ELSE (PICK SUCCESSFUL)
C
C
C*****        ADD 'IDSEAM' TO ARRAY OF PICKED SEAMS, 'IPICK'
C
                    if (.not.remove.and..not.direv.and..not.rmweld.and.
     *                 .not.mkpath.and.defwel) then
                      ICOUNT=ICOUNT+1
                      IPICK(ICOUNT)=IDSEAM
                      update=.true.
                   endif
C
C
                   call ringbe
                   write(unit=string,fmt='(i2)') idseam
                   call drtext('IDSEAM=   ',string)

c       Depending on the picking mode, do one of the following:

c              Remove the picked seam from the set of user-defined seams.

                   if (remove) then
                      call rmseam(idseam,grobjs)
                      remove=.false.

c              Reverse the direction of the picked seam.

                   elseif (direv) then
                      call revdir(idseam,sequen,seams,grobjs)
                      direv=.false.

c              'Unweld' the picked welded seam.

                   elseif (rmweld) then
                      call unweld(idseam,ipick,icount)
                      rmweld=.false.

c              Define an inter-seam path.

                   elseif (mkpath) then
                      do 99 index=1,icount
                         if(ipick(index).eq.idseam) idweld=index
99                    continue
                      mkpath=.false.
                      makep=.true.
                      start2=.true.


c              Copy the picked seam data and attempt to generate a
c              robot trajectory to weld the seam.

                   elseif (defwel) then
                      istart=sequen(1,idseam)
                      iend=istart+sequen(2,idseam)-1
                      nfiles=0
                      do 50 i=istart,iend
```

```
                        nfiles=nfiles+1
                        do 55 j=1,3
                           files(j,nfiles)=seams(j,i)
55                      continue
50                   continue

                     call genpat(idseam,files,nfiles,ipick,icount)

                     mvtabl=.false.


                  else

c            No picking mode is active.

                     call ringbe
                     call drtext('NO PICKING MODE IS ACTIVE',' ')

                  endif

               endif
            endif
         endif
      endif


      return
      end
```

```
        subroutine splsid(pair,result)

c       Test a pair of objects to see which one is 'behind' their separating
c       plane. The separating plane is specified in viewing coordinates.

c       PARAMETERS:
c               pair-   the pair of objects to be tested
c               result- .true. if pair(2) is 'behind' pair(1)

c       Called from:  ohtest,osort2

c       Variables:
c               icount- pointer into the separating plane list

c*******************************************************************************

$include seplane.com

        integer pair(2)
        logical result


        result=.false.

c       Find the correct separating plane from the list. The order of the
c       objects can be reversed to match the tabulated pairs.

        if (pair(1).gt.pair(2)) then
           icount=ptabl(pair(1))+pair(2)-1
           sign=1.
        else
           icount=ptabl(pair(2))+pair(1)-1
           sign=-1.
        endif


        dotprd=0.
        do 10 i=1,3
           dotprd=dotprd+vlist(i,icount)*nlist(i,icount)
10      continue

        dotprd=sign*dotprd

        if (dotprd.le.0.) result=.true.

        return
        end
        subroutine store(lunit,file,out,ipick,icount,seams,sequen,nseq)

c       Store and retrieve previous seam and weld definition data from a file.

c       Parameters:     lunit-  logical unit number
c                       file-   file name
c                       out-    .true. if seam definition to be stored
c                               .false. if seam definition to be retrieved

c                       ipick-  array of welded seams
c                       icount- number of welded seams
c                       seams-  array of segment pointers
c                       sequen- segment ids, and pointers into reference seam
c                               file.
c                       nseq-   number of segments in a seam definition.

c       Subroutines called: none

c       Called from: main

c       Files read: 'file'

c       Files written: 'file'

c*********************************************************************************

        integer ipick(*),seams(3,*),sequen(2,*),icount,nseq
        logical out
        character*(*) file


        if (out) then

c          Store seam definition in file.

           open(unit=lunit,file=file,status='new')
           write(lunit,*) icount
           do 10 i=1,icount
              write(lunit,*) ipick(i)
10         continue
           write(lunit,*) nseq
           do 15 ind=1,nseq
              write(lunit,*) (sequen(j,ind),j=1,2)
              istart=sequen(1,ind)
              iend=istart+sequen(2,ind)-1
              do 20 ind2=istart,iend
                 write(lunit,*) (seams(j,ind2),j=1,3)
20            continue
15         continue

        else

c          Retrieve seam definition from file.

           open(unit=lunit,file=file,status='old')
           read(lunit,*) icount
           do 30 i=1,icount
              read(lunit,*) ipick(i)
30         continue
           read(lunit,*) nseq
           do 35 ind=1,nseq
              read(lunit,*) (sequen(j,ind),j=1,2)
              istart=sequen(1,ind)
              iend=istart+sequen(2,ind)-1
              do 40 ind2=istart,iend
                 read(lunit,*) (seams(j,ind2),j=1,3)
40            continue
35         continue
        endif

        close(lunit)

        return
        end
```

```
      SUBROUTINE TANGEN(R,S,T,IFLAG)
C
C     THIS SUBROUTINE CALCULATES TANGENT OF CURVE, T.
C
C     INPUTS:
C           R(3,3)=POSITION VECTORS FOR POINTS I-1, I, I+1.
C           S(3)=ARC LENGTHS AT POINTS I-1, I, I+1.
C           IFLAG=CONDITION FLAG WHERE
C              I=1...SECOND POINT ON CURVE.
C              I=2...INTERMEDIATE POINTS.
C              I=3...LAST POINT ON CURVE.
C
C     OUTPUTS:
C           T(3)=TANGENT VECTOR AT POINT I.

c     Called from: proces

c     Subroutines called: unit

C
C*************************************************************************
C
C
      REAL R(3,3), S(3), T(3), TEMP(3)
C
C**  CASE OF (IFLAG)
C
      IF (IFLAG.LT.1.OR.IFLAG.GT.3) GO TO 400
      GO TO (100,200,300), IFLAG
C
C**  CASE (1)
C
100   CONTINUE
      DO 101 J=1,3
         TEMP(J)=(R(3,J)-R(2,J))/S(3)
101   CONTINUE
      GO TO 500
C
C**  CASE (2)
C
200   CONTINUE
      DO 102 J=1,3
         D1=R(3,1)-R(1,1)
         D2=R(3,2)-R(1,2)
         D3=R(3,3)-R(1,3)
         DS=SQRT(D1**2+D2**2+D3**2)
         TEMP(J)=(R(3,J)-R(1,J))/DS
102   CONTINUE
      GO TO 500
C
C**  CASE (3)
C
300   CONTINUE
      DO 103 J=1,3
         TEMP(J)=(R(2,J)-R(1,J))/(S(2)-S(1))
103   CONTINUE
```

```
      GO TO 500
C
C**  CASE ELSE
C
400   CONTINUE
      WRITE(*,150)
150   FORMAT(' CASE INDEX SPECIFICATION ERROR.')
C
500   CONTINUE

C***** NORMALIZE THE TANGENT VECTOR T.

      CALL UNIT(TEMP,T)

      RETURN
      END
      subroutine times(nwelds)

c     Calculate and store the time intervals between inter-weld positions
c     based on the maximum allowable joint velocities.

c     Parameters:
c             nwelds- the number of welds

c     Called from- main

c     Subroutines called- none

c     Variables:
c             nchain- number of kinematic chains
c             nrdof-  number of degrees of freedom
c             maxjv-  maximum allowable joint velocities
c             values- joint values
c             vlast-  previous joint values

c*******************************************************************************
c
c     File names.

      character*20 itable,ilocat

      real values(10,3),vlast(10,3)
      real maxjv(10,3)
      integer nchain,nrdof(3)

      integer recl(20)

      common /itable/itable
      common /ilocat/ilocat
      common /reclen/recl
      common /maxjv/maxjv
      common /nchain/nchain
      common /nrdof/nrdof


c     Open the inter-weld path files.

      open(unit=1,file=itable,form='unformatted',access='direct',
     *          status='old',recl=recl(7))
      open(unit=2,file=ilocat,form='unformatted',access='direct',
     *          status='old',recl=recl(8))


      if (nwelds.gt.0) then

      do 10 iseam=1,nwelds+1

c        Initialize the next weld sequence.

         read(1,rec=iseam) nstart,nend
         if (nstart.gt.0) then
```

```
c          Calculate the minimum time intervals between adjacent positions
c          along a path and store the result in the file.

           do 20 ipos=nstart,nend
              read(2,rec=ipos) (values(i,1),i=1,6),(values(i,2),i=1,3),time
              tmin=0.
              do 50 ind1=1,nchain
                 do 60 ind2=1,nrdof(ind1)
                    if (ipos.gt.nstart) then
                       temp=abs(values(ind2,ind1)-vlast(ind2,ind1))/
     *                    maxjv(ind2,ind1)
                       if (temp.gt.tmin) tmin=temp
                    endif
                    vlast(ind2,ind1)=values(ind2,ind1)
60               continue
50            continue
              write(2,rec=ipos) (values(i,1),i=1,6),(values(i,2),i=1,3),tmin
20         continue
           endif

10      continue

        endif

        return
        end
        subroutine tran(ivert1,ivert2,inorm1,inorm2)

c       Transform the vertex coords in the vertex list
c       from index 'ivert1' to index 'ivert2'
c       and transform the surface normals from index 'inorm1' to index 'inorm2'
c       by the current transformation.
c
c       CALLED FROM:
c               trandb
c
c       SUBROUTINES CALLED:
c               gtfnor
c               gtvcrd
c               tracrd
c               tranor
c
c       IRIS GRAPHICS CALLS:
c               getmat,loadma,transf,popmat,pushma
c
c       PARAMETERS:
C               ivert1- starting vertex
c               ivert2- ending vertex
c               inorm1- starting normal
c               inorm2- ending normal
c
c       VARIABLES:
c               postra(4)- homogeneous vector after transformation
c               pretra(4)-      ''         ''    before      ''
c               mat(4,4)-  transformation matrix
c
c*******************************************************************************

c

        real pretra(4),postra(4),mat(4,4)
        real preten(4,10),posten(4,10)

c       transform the range of vertices
c       If possible, transform blocks of ten.
c
        do 35 i=ivert1,ivert2
           if ((ivert2-i).ge.9) then
              call gtvcrd(10,i,preten)
              call transf(10,preten,posten)
              call tracrd(10,i,posten)
              i=i+9
           else
              call gtvcrd(1,i,pretra)
              call transf(1,pretra,postra)
              call tracrd(1,i,postra)
           endif
35      continue

c
c       transform the range of normals (rotations only)
c       If possible, transform blocks of ten.
```

```
c
      call pushma
      call getmat(mat)
      do 50 i=1,3
         mat(i,4)=0.
50    continue
      call loadma(mat)

      do 40 i=inorm1,inorm2
         if ((inorm2-i).ge.9) then
            call gtfnor(10,i,preten)
            call transf(10,preten,posten)
            call tranor(10,i,posten)
         else
            call gtfnor(1,i,pretra)
            call transf(1,pretra,postra)
            call tranor(1,i,postra)
         endif
40    continue

      call popmat

      return
      end
```

```
        SUBROUTINE TOLROT(ROTOLD,ICOUNT,ROTNEW,INHAG)
C
C       THIS SUBROUTINE CALCULATES AN INVERSE ROTATION MATRIX
C       FOR A ROTATION 'ROTNEW' ABOUT THE TOOL AXIS.
C       THE TOOL IS ROTATED ALTERNATELY BY AN ANGLE OF ICOUNT*DELTA
C       IN THE POSITIVE AND NEGATIVE DIRECTIONS.
C
C       INPUTS: ROTOLD- PREVIOUS ROTATION ANGLE
C               ICOUNT- NUMBER OF ROTATIONS
C
C       OUTPUTS: ROTNEW- NEW ROTATION ANGLE
C                INHAG(4,4)- INVERSE ROTATION MATRIX
C
c       Called from: worpos
C
C*******************************************************************************
C
        REAL INHAG(4,4),DELTA
        INTEGER IMAX
C
        COMMON /TOLROT/IMAX,DELTA
C
C*****  FIND THE NEW ROTATION ANGLE
C
        ROTNEW=ROTOLD+FLOAT((-1)**ICOUNT*ICOUNT/2)*DELTA
C
C*****  CALCULATE THE ROTATION MATRIX RELATING THE ROTATED TOOL TO THE
C*****  UNROTATED TOOL.
C
        INHAG(1,1)=COS(ROTNEW)
        INHAG(1,2)=SIN(ROTNEW)
        INHAG(2,1)=-INHAG(1,2)
        INHAG(2,2)=INHAG(1,1)
        INHAG(3,3)=1.
        INHAG(4,4)=1.
C
        RETURN
        END
        subroutine trandb(type,values)

c       Transform the geometric database specifying kinematic chains of
c       rigid links by performing a chain of transformations on each link.
c       The transformations are in a standard form, and are functions of
c       given joint variables.

c       PARAMETERS:
c               type-   0- perform only modelling transformation so that
c                          transformed points are in world coords.
c                       1- perform viewing transformation also, so that
c                          transformed points are in viewing coords.
c               values- the values of the joint variables.

c       SUBROUTINES CALLED:
c               fclist,getpts,getsep,ptvlst,tran,transp

c       GRAPHICS CALLS:
c               loadma,popmat,pushma,polarv,rotate,transl

c       Called from: drpath,intest,main,mkpath,posobs

c       Variables:

c               csplan- list of known separating planes
c               len,dist,alpha,theta- Denavit-Hartenberg specifications
c                       for the kinematic chains
c               jtype-  array specifying each joint as prismatic or revolute
c               link-   array specifying attachment of parts to link frames
c               nchain- the number of kinematic chains
c               nrdof-  the number of degrees of freedom for each kinematic
c                       chain
c               hpre1,hpre2-  transformations from world frame to zeroth link
c                             frame
c               hpost1,hpost2- transformations from last link frames to torch
c                              and positioner reference frames
c               ftran1,ftran2-  transformation from world frame to torch and
c                               positioner reference frames
c               workpc- the cluster id of the workpiece

c*******************************************************************************

$include seplane.com

        integer nchain
        integer nrdof(3)
        integer jtype(10,3)
        integer link(10,3)
        real len(10,3),dist(10,3),alpha(10,3),theta(10,3)
        real values(10,3)
        integer*2 rot,iangle
        real ident(4,4)
        real vdist
        integer*2 azim,inc,twist
        integer type
        real hpre1(4,4),hpost1(4,4),hpre2(4,4),hpost2(4,4)
        real ftran1(4,4),ftran2(4,4)
        real temp(4,4)
        integer workpc

        common /robot/link,jtype,alpha,len,theta,dist
        common /workpc/workpc
        common /nchain/nchain
        common /nrdof/nrdof
        common /view/vdist,azim,inc,twist
        common /trans/hpre1,hpost1,hpre2,hpost2
        common /ftrans/ftran1,ftran2
        common /ident/ident

        call pushma

        icount=1

c       Transform the part clusters constituting the links of all the
c       open kinematic chains in the system. Each link is related to the
c       previous link by the Denavit-Hartenberg transformation. The net
c       transformation is the product of these transformations.

        do 10 ind1=1,nchain

           call loadma(ident)
           if(type.eq.1) call polarv(vdist,azim,inc,twist)

c          Transform the table and robot world frames relative to the
c          workstation world frame.

           if (ind1.eq.1) then
              call multma(hpre1)
           elseif (ind1.eq.2) then
              call multma(hpre2)
           endif


c          Transform each link of the kinematic chain by the Denavit-Hartenberg
c          transformation.


           do 20 ind2=1,nrdof(ind1)
              if (jtype(ind2,ind1).eq.0) then

c       Revolute joint.

                 rot=ifix(values(ind2,ind1)*10.)
                 call rotate(rot,'z')
                 call transl(len(ind2,ind1),0,dist(ind2,ind1))
              else

c       Prismatic joint.

                 rot=ifix(theta(ind2,ind1))
                 rot=0
                 call rotate(rot,'z')
                 call transl(len(ind2,ind1),0,values(ind2,ind1))
              endif
              iangle=10*alpha(ind2,ind1)
              call rotate(iangle,'x')

              if (link(ind2,ind1).ne.0) then

c        Transform the parts geometry data by transforming the contiguous range
c        of vertices and face normals of the parts cluster attached to the
c        current frame.

                 iclust=link(ind2,ind1)
                 call fclist(iclust,inorm1,inorm2)
                 call ptvlst(iclust,ivert1,ivert2)

                 call tran(ivert1,ivert2,inorm1,inorm2)

c        Transform the known separating planes of the parts comprising the
c        current cluster.

                 call getpts(iclust,ipt1,numb)
                 do 222 ind3=ipt1,ipt1+numb-1
                    if (ind3.gt.1) then
                    call getsep(ind3,isep,imax)

c                   WHILE (csplan(icount).le.imax)

101                 continue
                    if (csplan(icount).gt.imax) go to 100

                       if (csplan(icount).ge.isep) then
                          call transp(icount)
                       endif
                       icount=icount+1

                    go to 101
100                 continue

c                   END WHILE

                    endif
222              continue

              endif
20         continue

c          Perform fixed transformations relative to the last link frame.

           if (ind1.eq.1) then
              call multma(hpost1)
           elseif (ind1.eq.2) then
              call multma(hpost2)

c             Transform the workpiece cluster, which is attached to the
c             positioner.
```

```
              if (workpc.gt.0) then
                 call fclist(workpc,inorml,inorm2)
                 call ptvlst(workpc,ivertl,ivert2)
                 call tran(ivertl,ivert2,inorml,inorm2)

c                Transform known separating planes of the workpiece cluster.

                 call getpts(workpc,iptl,numb)
                 do 322 ind3=iptl,iptl+numb-1
                    if (ind3.gt.1) then
                    call getsep(ind3,isep,imax)

c                   WHILE (csplan(icount).le.imax)

201                 continue
                    if (csplan(icount).gt.imax) go to 200

                       if (csplan(icount).ge.isep) then
                          call transp(icount)
                       endif
                       icount=icount+1

                    go to 201
200                 continue

c                   ENDWHILE

                    endif
322              continue
              endif

           endif


c          Store the resultant modelling transformation matrix for the
c          last link of the robot and positioner.
c          If necessary, postmultiply the current transformation by the inverse
c          of the viewing transformation.

           call pushma
           if (type.eq.1) then
              call getmat(temp)
              call loadma(ident)
              call rotate(azim,'z')
              call rotate(inc,'x')
              call rotate(twist,'z')
              call transl(0.,0.,vdist)
              call multma(temp)
           endif

c          Save the resultant robot gripper and table transformations.

           if (indl.eq.1) then
              call getmat(ftranl)
           elseif (indl.eq.2) then
```

```
                call getmat(ftran2)
             endif
             call popmat

10       continue


         call popmat

         return
         end
```

```
        SUBROUTINE TRANS(H,P1,P2)
C
C       THIS SUBROUTINE PERFORMS THE FOLLOWING HOMOGENEOUS
C       TRANSFORMATION.
C
C               P2(4)=H(4,4)*P1(4)
C
C       WHERE P1,P2 ARE 4X1 VECTORS AND H IS A 4X4 HOMOGENEOUS
C       COORDINATE TRANSFORMATION MATRIX.
C
C******************************************************************************
C
        REAL H(4,4),P1(3),P2(3)
C
c       Note: IRIS hardware transformation 'transf' doesn't work
c       inside the scope of a makeob


        DO 100 I=1,3
          P2(I)=H(I,1)*P1(1)+H(I,2)*P1(2)+H(I,3)*P1(3)+H(I,4)
100     CONTINUE

        P2(4)=1.

        RETURN
        END
        subroutine transp(icount)

c       Transform the given known separating plane by the current transformatio
c       matrix.

c       PARAMETERS:
c               icount- index of the separating plane to be transformed

c       Called from: trandb

c       GRAPHICS CALLS:
c               getma,loadma,popma,pushma,transf

c*******************************************************************************

$include seplane.com

        real pretra(4),postra(4),mat(4,4)

        index=csplan(icount)


c       Transform the plane vector.

        do 15 i=1,3
           pretra(i)=fvlist(i,icount)
15      continue
        pretra(4)=1.

        call transf(1,pretra,postra)
        do 30 i=1,3
           vlist(i,index)=postra(i)
30      continue


c       Transform the plane normal (rotation only).

        call pushma
        call getmat(mat)
        do 10 i=1,3
           mat(i,4)=0.
10      continue
        call loadma(mat)

        do 5 i=1,3
           pretra(i)=fnlist(i,icount)
5       continue
        pretra(4)=1.

        call transf(1,pretra,postra)

        do 20 i=1,3
           nlist(i,index)=postra(i)
20      continue

        call popmat
```

```
      return
      end
```

```
        subroutine unweld(idseam,ipick,icount)

c       Remove the weld trajectory for a picked seam.

c       Parameters:     idseam- the seam id
c                       ipick-  the ordered list of welded seams
c                       icount- the number of welded seams

c       Subroutines called: none

c       Called from: smpick

c       Parameters modified: ipick,icount

c       Common blocks modified: update

c       Files read: 'ptable', 'itable', 'ilocat'

c       Files written: 'ptable', 'itable', 'ilocat'

c*******************************************************************************


        integer ipick(*),icount,idseam
        character*20 ptable
        logical flag
        logical update
        character*20 itable,ilocat
        integer savend,nrdof(3)
        real save(10,3)
        integer recl(20)

        common /ptable/ptable
        common /update/update
        common /itable/itable
        common /ilocat/ilocat
        common /nrdof/nrdof
        common /reclen/recl


c       Find the weld trajectory corresponding to the picked seam. If no weld
c       is found, the picked seam has not yet been welded.

        flag=.false.
        do 10 i=1,icount
           if (.not.flag) then
              if (ipick(i).eq.idseam) then
                 flag=.true.
                 isave=i
              endif
           endif
10      continue


c       If it has been welded, remove the seam from the welded seam
```

```
c       list and from the locations
c       pointer table.

        if (flag) then
           open(unit=1,file=ptable,form='unformatted',access='direct',
     *          status='old',recl=recl(5))
           do 20 ind=isave,icount-1
              ipick(ind)=ipick(ind+1)
              read(1,rec=ind+2) nstart,nend
              write(1,rec=ind+1) nstart,nend
20         continue
           ipick(icount)=0
           icount=icount-1
           write(1,rec=1) icount
           write(1,rec=icount+2) 0,0
           close(1)

c       Revise the inter-weld paths so that the previous and following welds
c       are connected.

c          Revise the positions pointer table.

           open(unit=1,file=itable,form='unformatted',access='direct',
     *          status='old',recl=recl(7))

           do 55 i=1,icount+1
              read(1,rec=i) nstart,nend
55         continue

           read(1,rec=isave) istart,ntemp
           read(1,rec=isave+1) ntemp,nend
           ndiff=nend-istart-1
           nend=istart+1
           savend=nend
           write(1,rec=isave) istart,nend

           index=isave

30         continue
           index=index+1
           read(1,rec=index+1) nstart,nend
           if (nstart.ne.0) then
              nstart=nstart-ndiff
              nend=nend-ndiff
              savend=nend
              write(1,rec=index) nstart,nend
              goto 30
           endif

           write(1,rec=index) 0,0

           close(1)

c          Revise the inter-weld positions table 'ilocat'.
```

```
            open(unit=1,file=ilocat,form='unformatted',access='direct',
     *           status='old',recl=recl(8))

            do 40 ind=istart+1,savend

               read(1,rec=ind+ndiff) (save(i,1),i=1,nrdof(1)),
     *            (save(i,2),i=1,nrdof(2)),time

               write(1,rec=ind) (save(i,1),i=1,nrdof(1)),
     *            (save(i,2),i=1,nrdof(2)),time

40          continue

            close(1)

            update=.true.

         endif

         return
         end
```

```
      subroutine wgeom(idseam,files,nfiles)

c     Expert welder module which generates the geometric welding parameters
c     which are needed for generating a welding trajectory.  These parameters
c     are a function of the given geometric welding parameters only,
c     and are not a function of the weld orientation.

c     Subroutines called: exprtl

c     Called from: genpat

c     Variables:
c             jtype-  joint type
c             xsect-  weld cross sectional area
c             tl,t2-  thicknesses of the parts
c             icurve- straight or curved seam
c             speed-  welding speed
c             alpha-  torch longitudinal angle
c             beta-   torch transverse angle
c             stkout- torch gap

c*****************************************************************************

      integer idseam,files(3,*),nfiles
      integer recl(20)

      character*20 wpcdat,procad
      character*20 weltem

      common /exprt/volt,wsize,feed,gfr,walpha,beta,stkout
      common /wpcdat/wpcdat
      common /procad/procad
      common /weltem/weltem
      common /reclen/recl


c     Open the welding data files.

      open(unit=1,file=weltem,form='unformatted',status='new')
      open(unit=2,file=wpcdat,form='unformatted',access='direct',
     *        status='old',recl=recl(3))

c     Find the torch orientation, stick-out and welding speed as a function
c     of the weld x-section, type, and thicknesses.

      do 10 i=1,nfiles
         irec=files(1,i)
         read(2,rec=irec) jtype,xsect,tl,t2,icurve
         call exprtl(wsize,jtype,xsect,tl,t2,speed,alpha,beta,stkout)
         istart=files(2,i)
         iend=files(3,i)
         if (istart.gt.iend) then
            incr=-1
         else
            incr=1
         endif
```

```
            do 20 ind=istart,iend,incr
               write(1) speed,alpha,beta,stkout
20          continue
10       continue

         close(1)
         close(2)


         return
         end
         SUBROUTINE WORPOS(IDSEAM,files,nfiles,ISTAT)
C
c       This subroutine searches for a feasible welding trajectory for a
c       given seam.  The inverse kinematic solution is found for each point
c       along the seam, and this solution is tested against the constraints of
c       physical joint limits and interference.  Iterative searching is done
c       by rotating the torch about its axis, and by retrying the seam with
c       different starting robot configurations.  If a feasible trajectory is
c       found, the joint positions are stored in a file.
C
c       Parameters:
C               IDSEAM- SEAM IDENTIFICATION NUMBER
c               files-  seam segment ids, and pointers into reference CAD seam
c                       data file
c               nfiles- number of segments in 'files'
C               ISTAT- FEASIBILITY STATUS
C                               0=TRAJECTORY IS FEASIBLE
C                               1=ROBOT WORK ENVELOPE EXCEEDED
C                               2=ROBOT JOINT LIMIT EXCEEDED
c                               3=joint change too large
C
c       Called from-  config
C
C       SUBROUTINES CALLED -    HMULT
c                               intest
C                               JOINTS
C                               LIMITS
c                               mtable
C                               NOAP5
C                               TOLROT
C                               NXPROD

c       Variables:
c               ftran2- workpiece to world transform
c               hwc-    seam to workpiece transform
c               hcg-    torch to seam transform
c               hoc-    seam to world transform
c               hog-    torch to world transform
c               h06-    last link to base transform
c               h0a-    tool flange to base transform
c               h0g-    torch to base transform
c               h6g-    torch to last link transform
c               ho0-    base to world transformation
c               h0c-    seam to base transformation
c               inho0-  inverse of ho0
c               inh6a-  inverse of h6a, the tool flange to last link transform
c               interf- .true. if interference is detected
c               ijoint- array of joints which have exceeded their limits
c               minmov- minimum table joint increments
c               movtab- initial and final table positions
c               mvtabl- .true. if the table moves during the weld
c               R-      seam location vector
c               rprev-  previous R
c               T,A,B-  seam orientation vectors
c               repeat- .true. if the current solution is not feasible
c               rlast,tlast- previous robot, table joint values
```

```
c                rtheta- array of robot joint angles
c                stopw-  .true. to exit the subroutine unconditionally
c                toobig- .true. if the joint change between positions is too big
c                tstep-  table joint increment size
c                tvars-  table joint values
c                values- robot and positioner joint values
c
C
C****************************************************************************
C
C
      REAL HWC(4,4),HCG(4,4)
      REAL HOC(4,4),HOG(4,4)
      REAL R(3),T(3),A(3),B(3),X(3),Y(3),Z(3),
     *  RTHETA(10),INHOO(4,4),INH6A(4,4)
      real rprev(3)
      REAL THETA(10,3)
      REAL INHAG(4,4),H6G(4,4),H06(4,4),HOG(4,4)
      REAL HOA(4,4),DELTA
      REAL RMIN(10,3),RMAX(10,3)
      REAL HOO(4,4),HOC(4,4),ENORM(3),ELAT(3)
      real htemp(4,4)
      INTEGER IJOINT(10),IMAX,NRDOF(3)
      CHARACTER*20 WPCDAT
      CHARACTER*20 JOINT
      character*20 ptable,plocat,weltem,corect
      character*20 procad,cadlst
      character*20 itable,ilocat
      character*20 wptemp
      real temp1(4,4),temp2(4,4)
      real values(10,3)
      logical interf
      real ftran1(4,4),ftran2(4,4)
      real tvars(10)
      real movtab(10,2),tstep(10)
      logical mvtabl,tinit,tfinal
      integer files(3,*),nfiles
      real minmov(10,3)
      logical toobig
      real rlast(10),tlast(10)
      logical repeat
      logical stopw
      integer recl(20)
C
      common /jlimit/rmin,rmax
      common /nrdof/nrdof
      COMMON /INVERS/INHOO,INH6A
      common /ftrans/ftran1,ftran2
      COMMON /TOLROT/IMAX,DELTA
      COMMON /ROBPOS/HOG
      COMMON /JOINT/JOINT
      COMMON /VIEWPO/LEFT,RIGHT,BOTTOM,TOP
      common /values/values
      common /movtab/movtab,tstep
      common /flags2/mvtabl,tinit,tfinal
      common /procad/procad
```

```
        common /cadlst/cadlst
        common /wpcdat/wpcdat
        common /plocat/plocat
        common /ptable/ptable
        common /itable/itable
        common /ilocat/ilocat
        common /corect/corect
        common /weltem/weltem
        common /wptemp/wptemp
        common /minmov/minmov
        common /npoint/npoint
        common /stopw/stopw
        common /reclen/recl
C
C*****  INITIALIZE PARAMETERS
C
        ISOLN=0
        ISTAT=0
        ICOUNT=1
        ROTOLD=0.
        de2rad=.017453293
C
C
c       Open data files.

        open(unit=1,file=wpcdat,form='unformatted',access='direct',
     *      status='old',recl=recl(3))
        open(unit=2,file=cadlst,form='unformatted',access='direct',
     *      status='old',recl=recl(2))
        open(unit=3,file=ptable,form='unformatted',access='direct',
     *      status='old',recl=recl(5))
        open(unit=4,file=plocat,form='unformatted',access='direct',
     *      status='old',recl=recl(6))
        open(unit=7,file=corect,form='unformatted',access='direct',
     *      status='old',recl=recl(11))
        open(unit=5,file=procad,form='unformatted',access='direct',
     *      status='old',recl=recl(4))
        open(unit=6,file=weltem,form='unformatted',status='old')
        open(unit=8,file=joint,form='unformatted',status='new')
        open(unit=9,form='unformatted',status='scratch')
        open(unit=10,file=itable,form='unformatted',access='direct',
     *      status='old',recl=recl(7))
        open(unit=11,file=ilocat,form='unformatted',access='direct',
     *      status='old',recl=recl(8))
        open(unit=12,file=wptemp,form='unformatted',status='new')
        open(unit=13,form='unformatted',status='scratch')


c       Find the current end of the location file, and update the
c       pointer table.

        read(3,rec=1) nseams
        if (nseams.ge.1) then
           read(3,rec=nseams+1) nstart,nend
           nstart=nend+1
```

```
      else
         nstart=1
      endif
      nseams=nseams+1

      read(10,rec=nseams) lstart,lend
      if (nstart.eq.1) then
         write(11,rec=lstart) (values(i,1),i=1,nrdof(1)),
     *            (values(i,2),i=1,nrdof(2)),time
      endif
c
c     Calculate the table joint increments based on the given initial and
c     final positions, and the number of points on the seam.

      if (mvtabl) then
         do 10 i=1,nrdof(2)
            tstep(i)=(movtab(i,2)-movtab(i,1))/(npoint-1)
10       continue
      endif

c
c     Test each point on the seam for joint constraints and interference.
c
      index=0
      do 160 indl=1,nfiles
         if (.not.stopw) then
         istart=files(2,indl)
         iend=files(3,indl)
         if (istart.gt.iend) then
            incr=-1
         else
            incr=1
         endif
         do 150 ind=istart,iend,incr
            if (.not.stopw) then

            index=index+1
c
c*****      READ DATA FROM INPUT FILES.
c
            read(5,rec=ind) s,(r(i),i=1,3),(t(k),k=1,3),(a(l),l=1,3),
     *      (b(m),m=1,3)

            read(6) speed,walpha,beta,stkout

c           Calculate the time between positions to achieve the desired
c           welding speed.

            if (index.gt.1) then
               distan=0.
               do 110 i=1,3
                  distan=distan+((r(i)-rprev(i))**2)
110            continue
               distan=sqrt(distan)
               if (distan.gt.0.) then
```

```
                  time=distan/speed
               endif
            else
               time=0.
            endif
c
c           Save the time.
c
            write(13) time
c
c           Update the previous position.
c
            do 120 i=1,3
               rprev(i)=r(i)
120         continue
C
C
C*****      SET UP TRANSFORMATION MATRIX 'HWC'.  THE COLUMNS CONTAIN
C*****      THE VECTORS T, B, A, R, OBTAINED FROM 'procad'.
C
            HWC(1,1)=T(1)
            HWC(2,1)=T(2)
            HWC(3,1)=T(3)
            HWC(4,1)=0.
            HWC(1,2)=B(1)
            HWC(2,2)=B(2)
            HWC(3,2)=B(3)
            HWC(4,2)=0.
            HWC(1,3)=A(1)
            HWC(2,3)=A(2)
            HWC(3,3)=A(3)
            HWC(4,3)=0.
            HWC(1,4)=R(1)
            HWC(2,4)=R(2)
            HWC(3,4)=R(3)
            HWC(4,4)=1.
C
C*****      CONVERT ANGLES 'WALPHA', 'BETA' INTO RADIANS AND 'RHO'
C*****      INTO MILLIMETERS
C
            DALPHA=WALPHA*DE2RAD
            DBETA=BETA*DE2RAD
            DRHO=STKOUT
C
C*****      CALCULATE ELEMENTS OF MATRIX 'HCG'.
C
            X(1)=0.
            X(2)=COS(DALPHA)*COS(DBETA)
            X(3)=SIN(DBETA)
            Z(1)=-SIN(DALPHA)
            Z(2)=SIN(DBETA)
            Z(3)=-COS(DALPHA)*COS(DBETA)
            HCG(1,4)=-DRHO*Z(1)
            HCG(2,4)=-DRHO*Z(2)
            HCG(3,4)=-DRHO*Z(3)
            HCG(4,4)=1.
```

```
C
C*****        THE SECOND COLUMN IS THE CROSS PRODUCT OF THE THIRD AND
C*****               FIRST COLUMNS
C
              CALL NXPROD(Z,X,Y)
C
              DO 65 I=1,3
                 HCG(I,1)=X(I)
                 HCG(I,2)=Y(I)
                 HCG(I,3)=Z(I)
65            CONTINUE
C
              HCG(4,1)=0.
              HCG(4,2)=0.
              HCG(4,3)=0.
C
C
C*****        PERFORM MATRIX MULTIPLICATIONS TO OBTAIN HOG.
C*****        HOG=HOW*HWC*HCG (ftran2=HOW)
C
c             Find the joint values for the positioning table.

              do 1 i=1,nrdof(2)
                 if (mvtabl) then
                    tvars(i)=movtab(i,1)+(index-1)*tstep(i)
                 else
                    tvars(i)=values(i,2)
                 endif
1             continue

              call mtable(tvars)
              CALL HMULT(ftran2,HWC,HOC)
              CALL HMULT(HOC,HCG,HOG)

C
C
C
C*****        CALCULATE HOG=INHOO*HOG
C*****        'INHOO' IS THE INVERSE HOMOGENEOUS TRANSFORMATION RELATING
C*****        THE WORLD COORDINATE FRAME TO THE ROBOT BASE FRAME
C
              CALL HMULT(INHOO,HOG,HOG)
C
C

c       Search for a feasible solution.

895           CONTINUE

c              Check for a cancel command from the user.

               if (qtest().ne.0) then
                  ibut=qread(posit)
                  if (posit.ne.0) then
                     if (ibut.eq.pad0) then
                        stopw=.true.
                    endif
                 endif
              endif

             toobig=.false.

             if (.not.stopw) then
C
C
C*****       CALCULATE ROBOT JOINT ANGLES, TEST FOR LIMITS
C
C*****       IF THE ROBOT HAS 5 DEGREES OF FREEDOM, THEN CALCULATE
C*****       VECTORS 'N','O' AS FUNCTIONS OF VECTOR 'A' AND POSITION
C*****       'P'. THERE ARE TWO SOLUTIONS, SELECTED BY SETTING
C*****       'ISOLN'=0 OR 1. THE RESULT IS ONE OF TWO UNIQUE
C*****       TRANSFORMATIONS 'H0G'=[n,o,a,p]
C
                IF(NRDOF(1).EQ.5) CALL NOAP5(ISOLN)

C
C
C*****       SEARCH FOR A FEASIBLE SOLUTION BY ROTATING THE TOOL
C*****       ABOUT ITS AXIS. THIS APPLIES TO 6 DEGREE OF FREEDOM
C*****       ROBOTS ONLY, FOR WHICH THE TOOL FRAME VECTORS 'N','O'
C*****       CAN BE ARBITRARILY SPECIFIED. THE INITIAL 'N','O' ARE
C*****       AS CALCULATED PREVIOUSLY. THE FIRST CALL TO 'TOLROT'
C*****       YIELDS A ROTATION OF ZERO.
C
                CALL TOLROT(ROTOLD,ICOUNT,ROTNEW,INHAG)

C
C
C*****       CALCULATE ROBOT TRANSFORMATION MATRIX 'H06'.
C*****       H06=H0G*INHAG*INH6A
C
                CALL HMULT(H0G,INHAG,H0A)
                CALL HMULT(H0A,INH6A,H06)

C
C
C*****       CALCULATE JOINT ANGLES FOR A GIVEN ROBOT TOOL MOUNTING
C*****       FLANGE FRAME MATRIX 'H06'. IFLAG1: 0- OK, 1- ENVELOPE
C*****       EXCEEDED
C
                CALL JOINTS(H06,RTHETA,IFLAG1)

                repeat=.false.
                IF (IFLAG1.ne.0) then
                  repeat=.true.
                else

C
C
C*****       TEST THE CALCULATED JOINT POSITIONS AGAINST THEIR PHYSICAL
C*****       LIMITS. IFLAG2: 0- OK, 1- JOINT LIMIT EXCEEDED
C
                   CALL LIMITS(RTHETA,IFLAG2,IJOINT)
C
```

```
C
             IF(IFLAG2.ne.0) then
               repeat=.true.
             else

c    Test for interference

               call intest(rtheta,tvars,interf)
               if (interf) then
                 repeat=.true.
               else

c    Test for joint change limits.

                 if (index.gt.1) then

                   do 180 ind2=1,nrdof(1)
                     diff=abs(rtheta(ind2)-rlast(ind2))
                     if (diff.gt.minmov(ind2,1)) then
                       diff2=abs(diff-360.)
                       if (diff2.gt.minmov(ind2,1)) then
                         toobig=.true.
                       endif
                     endif
180                continue

                   if (.not.toobig) then
                     do 185 ind2=1,nrdof(2)
                       diff=abs(tvars(ind2)-tlast(ind2))
                       if (diff.gt.minmov(ind2,2)) then
                         diff2=abs(diff-360.)
                         if (diff2.gt.minmov(ind2,2)) then
                           toobig=.true.
                         endif
                       endif
185                  continue
                   endif

                 endif

                 if (toobig) then
                   iflag3=1
                   repeat=.true.
                 else

c        Update the previous angles.

                   do 190 ind2=1,nrdof(1)
                     rlast(ind2)=rtheta(ind2)
190                continue

                   do 195 ind2=1,nrdof(2)
                     tlast(ind2)=tvars(ind2)
195                continue
C
C
```

```
C*****        TEST JOINT 6 LIMITS (CHECK IF JOINT 6 ROTATES IN ONE
C*****        DIRECTION THROUGH A JOINT LIMIT). ALTHOUGH JOINT 6
C*****        ALLOWS MORE THAN 360. DEGREES OF ROTATION, WE CANNOT
C*****        DRIVE THE JOINT CONTINUOUSLY IN ONE DIRECTION. IF A
C*****        LIMIT IS REACHED, THE JOINT WILL ROTATE ONE ROTATION
C*****        AWAY FROM THE LIMIT BEFORE PROCEEDING. THIS CAN RESULT
C*****        IN A CATASTROPHIC DISCONTINUITY OF MOTION EVEN THOUGH
C*****        EACH LOCATION HAS A VALID SOLUTION.

c             The following calculations generate a true angle for joint 6.
c             The value returned from the kinematic calculations is always
c             in the range -180 to 180 degrees. One full rotation can be
c             added or subtracted, and the solution will be the same.
c             Thus, when joint 6 exceeds this range, we must add or subtract
c             one rotation from the calculated value. The joint 6 angles must
c             vary continuously between its limits.
C*****
C*****           T6LAST- LAST JOINT 6 ANGLE
C*****           ONEROT- ONE ROTATION (360. DEG.)
C
                             IF(index.EQ.1) THEN
C
C*****           INITIALIZE PARAMETERS AT START OF SEAM.
C
                                T6LAST=0.

                             ENDIF


c                Specify direction of one rotation.

                             aver=(rmin(6,1)+rmax(6,1))/2.
                             if (rtheta(6).lt.aver) then
                                onerot=360.
                             else
                                onerot=-360.
                             endif


c                Add one rotation if necessary to preserve the continuity
c                of the joint motion.

                             save1=rtheta(6)-t6last
                             save2=save1+onerot

                             IF (abs(SAVE2).LT.abs(SAVE1)) THEN
                                RTHETA(6)=RTHETA(6)+ONEROT
                             ENDIF

C
C*****        SAVE THE CURRENT JOINT POSITION

                             T6LAST=RTHETA(6)

C
C*****        IF 'RTHETA(6)' EXCEEDS JOINT LIMIT, repeat.
C
```

```
                              IF(RTHETA(6).LT.RMIN(6,1).OR.
     *                           RTHETA(6).GT.RMAX(6,1)) repeat=.true.
C
C
                           endif
                        endif
                     endif
                  endif


                  if (repeat) then
C
C
C*****        IF PATH NOT FEASIBLE, ROTATE TOOL AND TRY AGAIN UNTIL
C*****        ALL SPECIFIED ROTATIONS HAVE BEEN TRIED.
C*****        FOR A 5 DEGREE OF FREEDOM ROBOT, TEST THE TWO POSSIBLE
C*****        JOINT SOLUTIONS.
C
                     IF(NRDOF(1).EQ.5) THEN
C
C*****            'ISOLN'=0 OR 1 SELECTS THE TWO POSSIBLE 5 DEGREE OF
C*****            FREEDOM SOLUTIONS
C
                        ISOLN=ISOLN+1
                        IF(toobig.or.ISOLN.GT.1) then
                           if (iflag1.eq.1) goto 861
                           if (iflag2.eq.1) goto 850
                           if (iflag3.eq.1) goto 860
                        else
                           GO TO 895
                        endif

                     ELSEIF (NRDOF(1).EQ.6) THEN
C
C*****        FOR A 6 DEGREE OF FREEDOM ROBOT, INCREMENT THE TOOL AXIS
C*****        ROTATION COUNTER 'ICOUNT' AND RETRY, UNTIL A MAXIMUM
C*****        ROTATION IS REACHED.
C
                        ICOUNT=ICOUNT+1
                        if (toobig.and.icount.gt.1.or.icount.gt.imax) then
                           if (iflag1.eq.1) goto 861
                           if (iflag2.eq.1) goto 850
                           if (iflag3.eq.1) goto 860
                        else
                           goto 895
                        endif

                     ENDIF

                     toobig=.false.
C
                  endif
```

```
c*****          If a feasible solution was found,
c*****          save angles, H06,HOC in temporary files.

                   write(9) ((H06(I,J),J=1,4),I=1,4)
                   WRITE(9) ((HOC(I,J),J=1,4),I=1,4)
                   WRITE(8) (RTHETA(I),I=1,nrdof(1)),(tvars(j),j=1,nrdof(2))

c               Save the weld orientation for later calculation
c               of welding parameters.

                   call hmult(ftran2,hwc,htemp)
                   write(12) (htemp(j,1),j=1,3),(htemp(j,3),j=1,3)

C*****          RESET TOOL ROTATION COUNTER 'ICOUNT' AND REDEFINE THE
C*****          INITIAL ROTATION OF THE TOOL AS THE CURRENT ROTATION
C
                   ICOUNT=1
                   ROTOLD=ROTNEW

                endif
                endif
150          continue

             endif
160       continue



c         If a feasible path has been found, store the robot and table
c         locations in a file.

c         Rewind the temporary storage files.

          rewind(6)
          REWIND(9)
          rewind(8)
          rewind(13)

c         Read the location data from temporary files and store in
c         location file.

          if (.not.stopw) then
             DO 123 IND=1,NPOINT

c               Read the stored variables.

                READ(9) ((H06(I,J),J=1,4),I=1,4)
                READ(9) ((HOC(I,J),J=1,4),I=1,4)
                read(8) (rtheta(i),i=1,nrdof(1)),(tvars(j),j=1,nrdof(2))
                read(13) time
```

```
C
C*****        CALCULATE HOC=INHOO*HOC
C
              CALL HMULT(INHOO,HOC,HOC)
C
C*****        FIND SEAM LATERAL AND NORMAL DIRECTIONS RELATIVE TO ROBOT
C*****        COORDS.
C
              DO 115 I=1,3
                 ENORM(I)=HOC(I,3)
                 ELAT(I)=HOC(I,2)
115           CONTINUE
C
C
c             Write the robot and table joint values, and the time intervals
c             into the location file.  Write the corresponding normal and
c             lateral vectors into a separate file for use in real time
c             error correction.
C
              index=ind+nstart-1
              write(4,rec=index) (rtheta(i),i=1,nrdof(1)),
     *           (tvars(j),j=1,nrdof(2)),time

              if (ind.eq.1) write(11,rec=lend) (rtheta(i),i=1,nrdof(1)),
     *           (tvars(j),j=1,nrdof(2)),time

              WRITE(7,rec=index) (ENORM(I),I=1,3),(ELAT(J),J=1,3)
C
123         CONTINUE


c           Update the pointer tables.

            write(3,rec=1) nseams
            write(3,rec=nseams+1) nstart,index
            write(3,rec=nseams+2) 0,0

            write(10,rec=nseams) lstart,lend
            lstart=lend+1
            lend=lstart+1
            write(10,rec=nseams+1) lstart,lend
            write(10,rec=nseams+2) 0,0


c           Write the new location into the locations file.

            write(11,rec=lstart) (rtheta(i),i=1,nrdof(1)),
     *       (tvars(j),j=1,nrdof(2)),time
            read(11,rec=1) (rtheta(i),i=1,nrdof(1)),
     *       (tvars(j),j=1,nrdof(2)),time
            write(11,rec=lend) (rtheta(i),i=1,nrdof(1)),
     *       (tvars(j),j=1,nrdof(2)),time


         endif
C
```

```
C
C*****  RESET THE TOOL ROTATION ANGLE TO ZERO
C
        ROTOLD=0.
C
        GO TO 1000
C
C*****  ERROR MESSAGES
C
C*****  ROBOT ENVELOPE EXCEEDED
C
861     CONTINUE
           ISTAT=1
        GO TO 1000
C
C*****  JOINT LIMIT EXCEEDED
C
850     CONTINUE
           ISTAT=2
        GO TO 1000

c       Joint change too big.

860     continue
           istat=3
        goto 1000
C
C
C
1000    CONTINUE
C
C*****  CLOSE DATA FILES.
C
        close(1)
        close(2)
        close(3)
        close(4)
        close(5)
        close(6)
        close(7)
        close(8)
        close(9,status='delete')
        close(10)
        close(11)
        close(12)
        close(13,status='delete')
C
C
        RETURN
        END
        subroutine wparam(idseam,files,nfiles)

c       Expert welder module which calculates welding parameters which
c       are a function of the weld orientation.  These calculations must
c       be done after the weld trajectory is generated.

c       Parameters:
c               idseam- the
c               files-  seam segment id, starting and ending pointers
c               nfiles- number of seam segments

c       Subroutines called: exprt2

c       Called from: genpat

c       Variables:
c               T,A-    seam tangent, normal in world coordinates
c               jtype-  joint type
c               xsect-  weld cross sectional area (inch)
c               t1,t2-  thicknesses of parent parts (inch)
c               volt-   welding voltage
c               wsize-  wire diameter (inch)
c               feed-   wire feed rate
c               gfr-    gas flow rate

c*******************************************************************************

        integer idseam,files(3,*),nfiles

        character*20 wpcdat,wkstat,cadlst,procad,wpoint,wtable,ptable
        character*20 weltem
        character*20 wptemp
        real T(3),A(3)
        integer recl(20)

        common /exprt/volt,wsize,feed,gfr,walpha,beta,stkout
        common /currnt/currnt
        common /wpcdat/wpcdat
        common /wkstat/wkstat
        common /cadlst/cadlst
        common /procad/procad
        common /wpoint/wpoint
        common /wtable/wtable
        common /ptable/ptable
        common /wptemp/wptemp
        common /reclen/recl

c       Find the starting point of the weld in the location table.

        open(unit=1,file=ptable,form='unformatted',access='direct',
     *       status='old',recl=recl(5))
        read(1,rec=1) nseams
        if (nseams.ge.1) then
           read(1,rec=nseams+1) nstart,nend
        else
           nstart=1
```

```
      endif
      close(1)

c     Open the data files.

      open(unit=1,file=wpoint,form='unformatted',access='direct',
     *     status='old',recl=recl(9))
      open(unit=2,file=wtable,form='unformatted',access='direct',
     *     status='old',recl=recl(10))
      open(unit=3,file=wpcdat,form='unformatted',access='direct',
     *     status='old',recl=recl(3))
      open(unit=4,file=wptemp,form='unformatted',status='old')


c     For each weld segment, find the known geometric welding parameters.

      index=nstart
      do 10 i=1,nfiles
         irec=files(1,i)
         read(3,rec=irec) jtype,xsect,t1,t2,icurve
         istart=files(2,i)
         iend=files(3,i)
         if (istart.gt.iend) then
            incr=-1
         else
            incr=1
         endif

         if (index.gt.1) then
            read(1,rec=index-1) ipoint
            ipoint=ipoint+1
         else
            ipoint=1
         endif

c        For each point along a weld segment, retrieve the weld orientation
c        vectors and calculate the unknown welding parameters.
c        Record the welding parameters in the file 'wtable'.

         do 20 ind=istart,iend,incr
            write(1,rec=index) ipoint
            if (icurve.eq.1) then
               read(4) (T(j),j=1,3),(A(j),j=1,3)

               call exprt2(jtype,xsect,t1,t2,T,A)

               write(2,rec=ipoint) volt,feed,gfr,currnt
               ipoint=ipoint+1
            endif
            index=index+1
20       continue

         if (icurve.eq.0) then
            read(4) (T(j),j=1,3),(A(j),j=1,3)
            call exprt2(jtype,xsect,t1,t2,T,A)
            write(2,rec=ipoint) volt,feed,gfr,currnt
```

```
           endif

10      continue


        close(1)
        close(2)

        return
        end
```

APPENDIX V.2- GEM Source Listing

```
C  PROGRAM GEM
C
C  FUNCTION: The function of this program is create a datafile
C   with name of form confn.dat containing the workstation model
C   information in a format suitable for use directly in program
C   AUTOWELD in file GEOM.DAT.
C
C  I/O PARAMETERS: Program GEM requires user input from the
C  terminal, requested with prompts , to describe
C   a) program option
C   b) name of output file with name of form 'confn.dat'
C   c) name of auxiliary file with name of form 'ndat.dat'
C   d) cluster and part number of part to be entered
C   e) part name (currently only 'box' or 'cyl')
C   f) number of dimensions in part - 3 for 'box',
C      2 for 'cyl'
C   g) part dimensions - X,Y,Z for 'box' and
C      diameter,height for 'cyl'; in mm.
C   h) part reference frame origin vector relative to
C      cluster reference frame
C   i) part reference frame unit basis vectors relative
C      to cluster reference frame
C   j) continued program entry options- 'yes' or 'no'
C   k) separating plane parts cluster numbers  and part numbers
C   l) separating plane point and unit normal
C The output of the program is a file with name of form 'confn.dat'
C which can be directly copied into file GEOM.DAT for use by
C program AUTOWELD
C
C  SUBROUTINES CALLED: ETVAL,SVAL,QVAL,LAMVAL,PART
C
C  COMMON BLOCKS: none
C
C *********************************************************************
C
      CHARACTER *3 PNAME,FTYPE,ANS
      CHARACTER *9 FILENAME
      CHARACTER *8 DATAFILE
      INTEGER S,SS,ET,H,Q,QQ,QVP
      REAL NR,NRP,MU,NU,NN
      DIMENSION P(10,10,3),RO(10,10,30,3),ROP(30,3)
      DIMENSION NR(10,10,30,3),NRP(30,3),MU(30,30,3)
      DIMENSION NU(30,30,3),VV(300,3),NN(300,3),R(20)
      DIMENSION HT(10,10,4,4),KK(10,10),MM(10,10),JJ(10)
      DIMENSION XI(10,10,3),UP(10,10,3),ZE(10,10,3)
      DIMENSION N1(10),N2(10),N3(16),N4(16),NTM(20)
      NCL=10
      NPAR=10
      MMAX=30
      KMAX=30
C
C  DISPLAY MENU AND ENTER FUNCTION  AND FILENAME
C
      PRINT *,'       PROGRAM GEM MENU'
      PRINT *,'                                      '
      PRINT *,'PROGRAM FUNCTION',' SELECTION NUMBER'
      PRINT *,'                  '
      PRINT *,'CREATE NEW FILE','            1'
      PRINT *,'CORRECT OR ADD ','            2'
      PRINT *,'CLUSTER OR PART'
      PRINT *,'CORRECT OR ADD','             3'
      PRINT *,'SEPARATING PLANE'
      PRINT *,'         '
      PRINT *,'       '
      PRINT *,'ENTER SELECTION NUMBER'
      READ *,NS
      PRINT *,'ENTER NEW OR EXISTING FILENAME IN FORM "CONFN.DAT"'
      READ *,FILENAME
C
C  ENTER NDAT.DAT AND SETUP INPUT/OUTPUT LOGICAL UNITS
C
      PRINT *,'ENTER AUXILIARY DATAFILE IN FORM "NDAT.DAT"'
      READ *,DATAFILE
      NU1=8
      NU2=9
      FTYPE='old'
      IF (NS.EQ.1) FTYPE='new'
        open(unit=nu1,status=ftype,file=filename)
        open(unit=nu2,status=ftype,file=datafile,access='direct',
     *  form='unformatted',recl=50)
      INEW=0
C
C  SELECT CODE FROM FUNCTION
C
      GO TO (10,10,500),NS
 10   CONTINUE
C
C  PROMPT AND ENTER CLUSTER NUMBER
C
      PRINT *,'ENTER CLUSTER NUMBER'
      READ *,I
C
C  SET AND STORE II
C
      IREC=1
      IF ((NS.EQ.1).AND.(I.EQ.1)) GO TO 11
      READ(NU2,REC=IREC) II
      IF (I.LE.II) GO TO 12
      INEW=1
 11   II=I
      WRITE(NU2,REC=IREC) II
 12   CONTINUE
C
C  PROMPT AND ENTER PART NUMBER J
C
      PRINT *,'ENTER PART NUMBER'
      READ *,J
C
C  SET AND STORE JJ(I)
C
      IREC=1+I
      IF ((NS.EQ.1).AND.(J.EQ.1)) GO TO 13
      IF ((NS.EQ.2).AND.(INEW.EQ.1).AND.(J.EQ.1)) GO TO 13
      READ(NU2,REC=IREC) JJ(I)
      IF (J.LT.JJ(I)) GO TO 14
 13   JJ(I)=J
      WRITE(NU2,REC=IREC) JJ(I)
 14   CONTINUE
C
C  PROMPT AND ENTER PART NAME
C
      PRINT *,'ENTER PART NAME, 3 CHARACTERS'
      READ *,PNAME
C
C  PROMPT AND ENTER PART DIMENSIONS
C
      PRINT *,'ENTER NUMBER OF DIMENSIONS IN PART'
      READ *,NPM
      DO 16 IKI=1,NPM
      PRINT *,'ENTER VALUE FOR DIMENSION',' ',IKI,' IN MM. OR DEG.'
 16   READ *,R(IKI)
C
C  CALCULATE,USING SUBROUTINE PART,VALUES FOR MM,KK,RO,NR
C  AND STORE
C
      NFF=1
      CALL PART(NFF,PNAME,R,ROP,NRP,KA,MA,IEF,NV,NF,N1,
     1N2,N3,N4)
      DO 25 IK=1,3
      DO 18 K=1,KA
      RO(I,J,K,IK)=ROP(K,IK)
      IREC=1+NCL+2*NCL*NPAR+(I-1)*NPAR*KMAX*3+(J-1)
     1*KMAX*3+(K-1)*3+IK
 18   WRITE(NU2,REC=IREC) RO(I,J,K,IK)
      DO 20 M=1,MA
      NR(I,J,M,IK)=NRP(M,IK)
      IREC=1+NCL+2*NCL*NPAR+NCL*NPAR*KMAX*3
     1+(I-1)*NPAR*MMAX*3+(J-1)*MMAX*3+(M-1)*3+IK
 20   WRITE(NU2,REC=IREC) NR(I,J,M,IK)
 25   CONTINUE
      KK(I,J)=KA
      MM(I,J)=MA
      IREC=1+NCL+(I-1)*NPAR+J
      WRITE(NU2,REC=IREC) KK(I,J),PNAME
      IREC=1+NCL+NCL*NPAR +(I-1)*NPAR+J
      WRITE(NU2,REC=IREC) MM(I,J),PNAME
C
C  PROMPT,ENTER AND STORE HT
C
      HT(I,J,4,1)=0.
      HT(I,J,4,2)=0.
      HT(I,J,4,3)=0.
      HT(I,J,4,4)=1.
      PRINT *,'ENTER PX,PY,PZ IN MM.'
      READ *,P(I,J,1),P(I,J,2),P(I,J,3)
      HT(I,J,1,4)=P(I,J,1)
      HT(I,J,2,4)=P(I,J,2)
      HT(I,J,3,4)=P(I,J,3)
      PRINT *,'ENTER FIRST COLUMN OF HT,3 ENTRIES'
      READ *,HT(I,J,1,1),HT(I,J,2,1),HT(I,J,3,1)
      PRINT *,'ENTER SECOND COLUMN OF HT,3 ENTRIES'
      READ *,HT(I,J,1,2),HT(I,J,2,2),HT(I,J,3,2)
      PRINT *,'ENTER THIRD COLUMN OF HT,3 ENTRIES'
      READ *,HT(I,J,1,3),HT(I,J,2,3),HT(I,J,3,3)
      DO 30 IK=1,4
      DO 30 JK=1,4
      IREC=1+NCL+2*NCL*NPAR+NCL*NPAR*KMAX*3
     1+NCL*NPAR*MMAX*3+(I-1)*NPAR*16+(J-1)*16+(IK-1)*4+JK
      WRITE(NU2,REC=IREC) HT(I,J,IK,JK)
 30   CONTINUE
C
C  CHECK FOR MORE PARTS
C
      PRINT *,'DO YOU WISH TO ENTER/CORRECT MORE PARTS-YES,NO?'
      READ *,ANS
      IF (ANS.EQ.'yes') GO TO 12
C
C  CHECK FOR MORE CLUSTERS
C
      PRINT *,'DO YOU WISH TO ENTER/CORRECT MORE CLUSTERS-YES,NO?'
      READ *,ANS
      IF (ANS.EQ.'yes') GO TO 10
 500  SS=0
      IREC=1+NCL+2*NCL*NPAR+NCL*NPAR*KMAX*3
     1+NCL*NPAR*MMAX*3+NCL*NPAR*16+150
      IF (NS.NE.1) READ(NU2,REC=IREC) SS
C
C READ IN II,JJ,KK,MM,HT,RO,NR FROM DISK FILE
C
      IREC=1
      READ(NU2,REC=IREC) II
      DO 400 I=1,II
      IREC=1+I
 400  READ(NU2,REC=IREC) JJ(I)
      DO 410 I=1,II
      DO 410 J=1,JJ(I)
      IREC=1+NCL+(I-1)*NPAR+J
      READ(NU2,REC=IREC) KK(I,J),PNAME
      IREC=1+NCL+NCL*NPAR+(I-1)*NPAR+J
      READ(NU2,REC=IREC) MM(I,J),PNAME
      DO 405 IK=1,4
      DO 405 JK=1,4
      IREC=1+NCL+2*NCL*NPAR+NCL*NPAR*KMAX*3
     1+NCL*NPAR*MMAX*3+(I-1)*NPAR*16+(J-1)*16+(IK-1)*4+JK
 405  READ(NU2,REC=IREC) HT(I,J,IK,JK)
      DO 409 IK=1,3
      DO 406 K=1,KK(I,J)
      IREC=1+NCL+2*NCL*NPAR+(I-1)*NPAR*KMAX*3
     1+(J-1)*KMAX*3+(K-1)*3+IK
 406  READ(NU2,REC=IREC) RO(I,J,K,IK)
      DO 407 M=1,MM(I,J)
      IREC=1+NCL+2*NCL*NPAR+NCL*NPAR*KMAX*3
     1+(I-1)*NPAR*MMAX*3+(J-1)*MMAX*3+(M-1)*3+IK
 407  READ(NU2,REC=IREC) NR(I,J,M,IK)
 409  CONTINUE
 410  CONTINUE
C
C CHECK IF USER WISHES TO ENTER/CORRECT SEPARATING PLANES
C
      PRINT *,'DO YOU WISH TO ENTER/CORRECT SEP. PLANES-YES,NO?'
      READ *,ANS
      IF (ANS.EQ.'no') GO TO 700
 501  CONTINUE
C
C  PROMPT AND ENTER CLUSTER AND PART NUMBERS FOR
C  FIRST AND SECOND PARTS FOR SEPARATING PLANES
C
      PRINT *,'KNOWN SEPARATING PLANE ENTRY'
      PRINT *,'FIRST PART MUST HAVE LOWEST CLUSTER NUMBER'
      PRINT *,'ENTER CLUSTER NUMBER,PART NUMBER OF FIRST PART'
      READ *,IP1,JP1
      PRINT *,'ENTER CLUSTER NUMBER,PART NUMBER OF SECOND PART'
      READ *,IP2,JP2
      SS=1+SS
C
C  CALCULATE L,N,S
C  VALUE FOR L
C
      IF (IP1.EQ.1) GO TO 510
      L=JP1
      DO 508 IP=1,IP1-1
 508  L=L+JJ(IP)
      GO TO 511
 510  L=JP1
 511  CONTINUE
C
C  VALUE FOR N
C
      IF (IP2.EQ.1) GO TO 513
      N=JP2
      DO 512 IP=1,IP2-1
 512  N=N+JJ(IP)
      GO TO 514
 513  N=JP2
 514  CONTINUE
C
C  VALUE FOR S
C
      IF (N.EQ.1) GO TO 517
      S=L
      DO 516 NPL=1,N-1
 516  S=S+NPL-1
      GO TO 518
 517  S=N-L
 518  CONTINUE
C
C  PROMPT AND ENTER SEPARATING PLANE POINT AND NORMAL
C
      PRINT *,'ENTER THREE COMPONENTS OF SEPARATING PLANE'
      PRINT *,'POINT SEPARATED BY COMMAS,RELATIVE TO '
      PRINT *,'CLUSTER REFERENCE FRAME OF SECOND PART'
      READ *,MU(L,N,1),MU(L,N,2),MU(L,N,3)
      PRINT *,'NOW ENTER IN SAME FORMAT THREE COMPONENTS'
      PRINT *,'OF SEPARATING PLANE UNIT NORMAL'
      READ *,NU(L,N,1),NU(L,N,2),NU(L,N,3)
C
C  STORE MU AND NU
C
      IREC=1+NCL+2*NCL*NPAR+NCL*NPAR*KMAX*3
     1+NCL*NPAR*MMAX*3+NCL*NPAR*16+SS
      WRITE(NU2,REC=IREC) S,L,N,(MU(L,N,IJ),IJ=1,3),(NU(L,N,IJ),IJ=1,3)
      IREC=IREC-SS+150
      WRITE(NU2,REC=IREC) SS
C
C  CHECK FOR MORE SEPARATING PLANE ADDITIONS/CORRECTIONS
C
      PRINT *,'WISH TO ENTER/CORRECT MORE SEPARATING PLANES?-YES,NO'
      READ *,ANS
      IF (ANS.EQ.'yes') GO TO 501
 700  CONTINUE
C
C  CALCULATE INDEX LIMITS H,QQ,LMC,SS
C
      H=0
      DO 710 I=1,II
 710  H=H+JJ(I)
      QQ=0
      DO 712 I=1,II
      DO 712 J=1,JJ(I)
 712  QQ=QQ+KK(I,J)
      LMC=0
      DO 714 I=1,II
      DO 714 J=1,JJ(I)
 714  LMC=LMC+MM(I,J)
C
C  CALCULATE VERTICES VV
C
      Q=1
      DO 725 I=1,II
      DO 725 J=1,JJ(I)
      DO 725 K=1,KK(I,J)
      DO 718 IK=1,3
 718  VV(Q,IK)=HT(I,J,IK,4)
      DO 720 IR=1,3
      DO 719 IC=1,3
 719  VV(Q,IR)=VV(Q,IR)+HT(I,J,IR,IC)*RO(I,J,K,IC)
 720  CONTINUE
      Q=Q+1
 725  CONTINUE
C
C  CALCULATE NORMALS NN
C
      LS=1
      DO 735 I=1,II
      DO 735 J=1,JJ(I)
      DO 735 M=1,MM(I,J)
      DO 728 IK=1,3
 728  NN(LS,IK)=0.
      DO 730 IR=1,3
      DO 729 IC=1,3
 729  NN(LS,IR)=NN(LS,IR)+HT(I,J,IR,IC)*NR(I,J,M,IC)
 730  CONTINUE
      LS=LS+1
 735  CONTINUE
C
C RETRIEVE SS
C
      IREC=1+NCL+2*NCL*NPAR+NCL*NPAR*KMAX*3
     1+NCL*NPAR*MMAX*3+NCL*NPAR*16+150
      READ(NU2,REC=IREC) SS
C
C  PLACE DATA IN PROPER FORMAT IN FILE CONFN.DAT
C
      WRITE(NU1,*) 'clusters CLUSTR'
      WRITE(NU1,*) 'no. of clusters'
      WRITE(NU1,*) 'clust. ID,first part,no.parts,first face,no.faces,
     1first vert,no.verts'
      WRITE(NU1,*) II
      DO 750 I=1,II
      CALL ETVAL(JJ,I,1,ET)
      CALL QVAL(KK,JJ,I,1,1,Q)
      NVERT=0
      JUP=JJ(I)
      DO 738 JP=1,JUP
 738  NVERT=NVERT+KK(I,JP)
      CALL LAMVAL(MM,JJ,I,1,1,LS)
      NFACE=0
      JUP=JJ(I)
      DO 740 JP=1,JUP
 740  NFACE=NFACE+MM(I,JP)
 750  WRITE(NU1,*) I,ET,JJ(I),LS,NFACE,Q,NVERT
      WRITE(NU1,*) 'PARTS'
      WRITE(NU1,*) 'no. of parts'
      WRITE(NU1,*) 'part ID,first face,no, of faces'
      WRITE(NU1,*) H
      DO 770 I=1,II
      DO 770 J=1,JJ(I)
      CALL ETVAL(JJ,I,J,ET)
      CALL LAMVAL(MM,JJ,I,J,1,LS)
 770  WRITE(NU1,*) ET,LS,MM(I,J)
      WRITE(NU1,*) 'known separating planes CSPLAN'
      WRITE(NU1,*) 'no. of separating planes'
      WRITE(NU1,*) 'index,part ID,other part ID,separating face ID'
      WRITE(NU1,*) SS
      IRECS=1+NCL+2*NCL*NPAR+NCL*NPAR*KMAX*3+NCL*NPAR*MMAX*3
     1+NCL*NPAR*16
      DO 775 IP=1,SS
      IREC=IRECS+IP
      READ(NU2,REC=IREC) S,L,N,(MU(L,N,IJ),IJ=1,3),(NU(L,N,IJ),IJ=1,3)
 775  WRITE(NU1,*) IP,N,L,S
      WRITE(NU1,*) 'separating planes'
      WRITE(NU1,*) 'index,vector,normal'
      DO 778 IP=1,SS
      IREC=IRECS+IP
      READ(NU2,REC=IREC) S,L,N,(MU(L,N,IJ),IJ=1,3),(NU(L,N,IJ),IJ=1,3)
  778 WRITE(NU1,*) S,(MU(L,N,IJ),IJ=1,3),(NU(L,N,IJ),IJ=1,3)
      WRITE(NU1,*) 'number of faces NFACE'
      NFACE=0
      DO 782 I=1,II
      DO 782 J=1,JJ(I)
  782 NFACE=NFACE+MM(I,J)
      WRITE(NU1,*) NFACE
      WRITE(NU1,*) 'vertex list VERTS'
      WRITE(NU1,*) 'number of vertex pointers'
      WRITE(NU1,*) 'face ID,no. of vertices,vertex IDs'
      NMULT=0
      DO 790 I=1,II
      DO 790 J=1,JJ(I)
      IREC=1+NCL+NCL*NPAR+(I-1)*NPAR+J
      READ(NU2,REC=IREC) MM(I,J),PNAME
      DO 790 M=1,MM(I,J)
      CALL PART(2,PNAME,R,ROP,NRP,KA,MA,IEF,1,M,N1,
     1N2,N3,N4)
  790 NMULT=NMULT+N4(1)
      WRITE(NU1,*) NMULT
      DO 800 I=1,II
      DO 800 J=1,JJ(I)
      IREC=1+NCL+NCL*NPAR+(I-1)*NPAR+J
      READ(NU2,REC=IREC) MM(I,J),PNAME
      DO 800 M=1,MM(I,J)
      CALL PART(2,PNAME,R,ROP,NRP,KA,MA,IEF,1,M,N1,
     1N2,N3,N4)
      CALL LAMVAL(MM,JJ,I,J,M,LSP)
      DO 792 IDD=1,N4(1)
      IND=1+IDD
      KV=N4(IND)
      CALL QVAL(KK,JJ,I,J,KV,Q)
  792 NTM(IDD)=Q
  800 WRITE(NU1,*) LSP,N4(1),(NTM(IP),IP=1,N4(1))
      WRITE(NU1,*) 'vertex coordinates VCOORD'
      WRITE(NU1,*) 'no. of vertices'
      WRITE(NU1,*) 'vertex ID,coordinates X,Y,Z'
      WRITE(NU1,*) QQ
      DO 810 Q=1,QQ
  810 WRITE(NU1,*) Q,(VV(Q,IK),IK=1,3)
      WRITE(NU1,*) 'faces adjacent to vertices,FACES2'
      WRITE(NU1,*) 'no. of adjacent face pointers'
      WRITE(NU1,*) 'vertex ID,no. of adjacent faces,face IDs'
      NMULT=0
      DO 820 I=1,II
      DO 820 J=1,JJ(I)
      IREC=1+NCL+(I-1)*NPAR+J
      READ(NU2,REC=IREC) KK(I,J),PNAME
      DO 820 K=1,KK(I,J)
      CALL PART(2,PNAME,R,ROP,NRP,KA,MA,IEF,K,1,N1,
     1N2,N3,N4)
  820 NMULT=NMULT+N1(1)
      WRITE(NU1,*) NMULT
      DO 830 I=1,II
      DO 830 J=1,JJ(I)
      IREC=1+NCL+(I-1)*NPAR+J
      READ(NU2,REC=IREC) KK(I,J),PNAME
      DO 830 K=1,KK(I,J)
      CALL PART(2,PNAME,R,ROP,NRP,KA,MA,IEF,K,1,N1,N2,N3,N4)
      CALL QVAL(KK,JJ,I,J,K,Q)
      QVP=Q
      DO 822 IDD=1,N1(1)
      IND=1+IDD
      MVL=N1(IND)
      CALL LAMVAL(MM,JJ,I,J,MVL,LS)
  822 NTM(IDD)=LS
  830 WRITE(NU1,*) QVP,N1(1),(NTM(IP),IP=1,N1(1))
      WRITE(NU1,*) 'faces adjacent to faces,FACES3'
      WRITE(NU1,*) 'no. of face pointers'
      WRITE(NU1,*) 'face id,no. of adjacent faces,face IDs'
      NMULT=0
      DO 840 I=1,II
      DO 840 J=1,JJ(I)
      IREC=1+NCL+NCL*NPAR+(I-1)*NPAR+J
      READ(NU2,REC=IREC) MM(I,J),PNAME
      DO 840 M=1,MM(I,J)
      CALL PART(2,PNAME,R,ROP,NRP,KA,MA,IEF,1,M,N1,N2,N3,N4)
  840 NMULT=NMULT+N3(1)
      WRITE(NU1,*) NMULT
      DO 850 I=1,II
      DO 850 J=1,JJ(I)
      IREC=1+NCL+NCL*NPAR+(I-1)*NPAR+J
      READ(NU2,REC=IREC) MM(I,J),PNAME
      DO 850 M=1,MM(I,J)
      CALL PART(2,PNAME,R,ROP,NRP,KA,MA,IEF,1,M,N1,
     1N2,N3,N4)
      CALL LAMVAL(MM,JJ,I,J,M,LSP)
      DO 842 IDD=1,N3(1)
      IND=1+IDD
      MVL=N3(IND)
      CALL LAMVAL(MM,JJ,I,J,MVL,LS)
  842 NTM(IDD)=LS
  850 WRITE(NU1,*) LSP,N3(1),(NTM(IP),IP=1,N3(1))
      WRITE(NU1,*) 'adjacent vertices ADJVRT'
      WRITE(NU1,*) 'no. of adjacent vertex pointers'
      WRITE(NU1,*) 'vertex ID,no. of adjacent vertices,adjacent
     1vertex IDs'
      NMULT=0
      DO 860 I=1,II
      DO 860 J=1,JJ(I)
      IREC=1+NCL+(I-1)*NPAR+J
      READ(NU2,REC=IREC) KK(I,J),PNAME
      DO 860 K=1,KK(I,J)
      CALL PART(2,PNAME,R,ROP,NRP,KA,MA,IEF,K,1,N1,
     1N2,N3,N4)
  860 NMULT=NMULT+N2(1)
      WRITE(NU1,*) NMULT
      DO 870 I=1,II
      DO 870 J=1,JJ(I)
      IREC=1+NCL+(I-1)*NPAR +J
      READ(NU2,REC=IREC) KK(I,J),PNAME
      DO 870 K=1,KK(I,J)
      CALL PART(2,PNAME,R,ROP,NRP,KA,MA,IEF,K,1,
     1N1,N2,N3,N4)
      CALL QVAL(KK,JJ,I,J,K,Q)
      QVP=Q
      DO 862 IDD=1,N2(1)
      IND=1+IDD
      KV=N2(IND)
      CALL QVAL(KK,JJ,I,J,KV,Q)
 862  NTM(IDD)=Q
 870  WRITE(NU1,*) QVP,N2(1),(NTM(IP),IP=1,N2(1))
      WRITE(NU1,*) 'face normals FNORMS'
      WRITE(NU1,*) 'no. of faces'
      WRITE(NU1,*) 'face ID,unit normal X,Y,Z'
      WRITE(NU1,*) LMC
      DO 880 LS=1,LMC
 880  WRITE(NU1,*) LS,(NN(LS,IP),IP=1,3)
      STOP
      END
C
      SUBROUTINE ETVAL(JJ,I,J,ET)
C
C  FUNCTION: The subroutine determines the overall part
C  number for a specified part in a cluster
C
C  INPUT ARGUMENTS: JJ(I)=number of parts in Ith cluster
C                   I=cluster number
C                   J=part number within cluster
C
C  OUTPUT ARGUMENTS: ET=overall part number
C
C  SUBROUTINES CALLED: none
C
C  CALLED FROM: main GEM
C
C  COMMON BLOCKS: none
C
C *************************************************************
C
      INTEGER ET
      DIMENSION JJ(10)
      IF (I.EQ.1) GO TO 50
      ET=J
      DO 10 IP=1,I-1
 10   ET=ET+JJ(IP)
      RETURN
 50   ET=J
      RETURN
      END
C
      SUBROUTINE QVAL(KK,JJ,I,J,K,Q)
C
C  FUNCTION: This subroutine determines the overall vertex
C  number for a given vertex on a part in a cluster
```

```
C
C  INPUT ARGUMENTS: KK(I,J)=total number of vertices on Jth part
C                           of Ith cluster
C                   JJ(I)=number of parts in Ith cluster
C                   I=cluster number
C                   J=part number in Ith cluster
C                   K=vertex number relative to Jth part of
C                     Ith cluster
C
C  OUTPUT ARGUMENTS: Q=overall vertex number
C
C  SUBROUTINES CALLED: none
C
C  CALLED FROM: main GEM
C
C  COMMON BLOCKS: none
C
C *****************************************************************
C
      INTEGER Q
      DIMENSION KK(10,10),JJ(10)
      Q=K
      IF (J.EQ.1) GO TO 50
      DO 20 JP=1,J-1
   20 Q=Q+KK(I,JP)
   50 IF (I.EQ.1) RETURN
      DO 70 IP=1,I-1
      DO 70 JP=1,JJ(IP)
   70 Q=Q+KK(IP,JP)
      RETURN
      END
C
      SUBROUTINE LAMVAL(MM,JJ,I,J,M,LS)
C
C  FUNCTION: This subroutine determines the overall face number for
C  a given face on part in a cluster
C
C  INPUT ARGUMENTS: MM(I,J)=total number of faces on Jth part of
C                           Ith cluster
C                   JJ(I)=total number of parts in Ith cluster
C                   I=cluster number
C                   J=part number in Ith cluster
C                   M=face number on Jth part in Ith cluster
C
C  OUTPUT ARGUMENTS: LS=overall face number
C
C  SUBROUTINES CALLED: none
C
C  CALLED FROM: main GEM
C
C  COMMON BLOCKS: none
C
C ****************************************************************
C
      DIMENSION MM(10,10),JJ(10)
      LS=M
      IF (J.EQ.1) GO TO 50
      DO 20 JP=1,J-1
 20   LS=LS+MM(I,JP)
 50   IF (I.EQ.1) RETURN
      DO 70 IP=1,I-1
      DO 70 JP=1,JJ(IP)
 70   LS=LS+MM(IP,JP)
      RETURN
      END
C
      SUBROUTINE PART(NFF,NAME,R,ROP,NRP,KA,MA,IEF,NV,NF,N1,
     1N2,N3,N4)
C
C  FUNCTION: This subroutine determines , for a given part name
C  and dimensions , the number of part vertices, faces, the
C  identifying postion and normal vectors for vertices and faces,
C  and as well relational information for adjoining faces and
C  vertices.
C
C  INPUT ARGUMENTS: NFF= function selection: 1 to find number of
C                            vertices and faces and vertex postion
C                            and face normal vectors ; 2 to find
C                            adjoining face and vertex relational
C                            information
C                   NAME=part name: only two part names currently
C                            recognized i.e. 'box' or 'cyl'
C                   R= vector containing part dimensions ;
C                         for box , R(1),R(2),R(3) are X,Y,Z of box
C                         in mm. ; for cyl R(1),R(2) are diameter
C                         and height in mm.
C                   NV=vertex number in part,used when NFF=2
C                   NF=face number in part,used when NFF=2
C
C  OUTPUT ARGUMENTS: ROP= matrix containing vertices position
C                             vectors
C                    NRP= matrix containing faces normal vectors
C                    KA=number of vertices in part
C                    MA=number of faces in part
C                    IFF=error code ; IEF=1 if name in error
C                    N1= vector containing N1(1)=no . of faces
C                       adjacent to vertex NV and N1(I),I=2,1+N1(1)
C                       the face numbers of those adjacent faces.
C                    N2= vector containing N2(1)=no. of vertices
C                         adjacent to vertex NV and N2(I),I=2,1+N2(1)
C                         the vertex numbers of these vertices.
C                    N3= vector containing N3(1)=no. of faces
C                          adjacent to face NF and N3(I),I=2,N3(1)+1
C                         the faced numbers of these faces
C                    N4= vector containing N4(1)=no. of vertices
C                         adjacent to face NF and N4(I),I=2,N4(1)+1
C                         the vertex numbers of these vertices.
C
C  SUBROUTINES CALLED: none
C
C  CALLED FROM: main GEM
C
```

```
C  COMMON BLOCKS: none
C
C ***********************************************************************
C
      CHARACTER *3 NAME
      REAL NRP(30,3)
      DIMENSION R(20),ROP(30,3)
      DIMENSION N1(10),N2(10),N3(16),N4(16)
C
C  SET ERROR FLAG OFF
C
      IDP=9999
      IEF=0
C
C  SELECT FUNCTION
C
      IF (NFF.EQ.2) GO TO 1000
C
C  EXAMINE PART NAME AND ASSIGN PART ID
C
      IF (NAME.EQ.'box') IDP=1
      IF (NAME.EQ.'cyl') IDP=2
C
C  VALID NAME?
C
      IF (IDP.EQ.9999) GO TO 999
C
C  SELECT SECTION OF CODE FOR PART ID
C
      GO TO (10,100),IDP
C
C  ASSIGN INDEX K,M LIMITS
C
 10   KA=8
      MA=6
C
C  READ DIMENSIONS
C
      X=R(1)
      Y=R(2)
      Z=R(3)
C
C  ASSIGN VERTICES POSITION VECTORS AND FACE NORMALS
C
      DO 15 II=1,3
      DO 13 M=1,MA
 13   NRP(M,II)=0.
      DO 14 K=1,KA
 14   ROP(K,II)=0.
 15   CONTINUE
      ROP(2,1)=X
      ROP(3,1)=X
      ROP(6,1)=X
      ROP(7,1)=X
      ROP(3,2)=Y
      ROP(4,2)=Y
      ROP(7,2)=Y
      ROP(8,2)=Y
      ROP(5,3)=Z
      ROP(6,3)=Z
      ROP(7,3)=Z
      ROP(8,3)=Z
      NRP(1,2)=-1.
      NRP(2,1)=1.
      NRP(3,2)=1.
      NRP(4,1)=-1.
      NRP(5,3)=-1.
      NRP(6,3)=1.
C
C  RETURN WITH NO NAME ERROR
C
      RETURN
 100  CONTINUE
C
C  READ NUMBER OF FACES
C
      PRINT *,'ENTER NUMBER OF CYLINDER WALL SEGMENTS,MAXIMUM OF 15'
      READ *,NSEG
C
C ASSIGN INDEX K,M LIMITS
C
      KA=2*NSEG
      MA=NSEG+2
C
C  READ DIMENSIONS
C
      D=R(1)
      H=R(2)
C
C  ASSIGN VERTICES POSITION VECTORS
C  AND FACE NORMALS
C
      DO 115 II=1,3
      DO 113 M=1,MA
 113  NRP(M,II)=0.
      DO 114 K=1,KA
 114  ROP(K,II)=0.
 115  CONTINUE
      DTHETA=2.*3.14159/NSEG
      THETA=0.
      NRP(1,3)=-1
      NRP(2,3)=1
      DO 118 I=1,NSEG
      THETA=THETA+DTHETA
      ROP(I,3)=0.
      ROP(I+NSEG,3)=H
      ROP(I,1)=COS(THETA)*D/2.
      ROP(I+NSEG,1)=ROP(I,1)
      ROP(I,2)=SIN(THETA)*D/2.
      ROP(I+NSEG,2)=ROP(I,2)
      THAV=THETA-DTHETA/2.
      NRP(I+2,1)=COS(THAV)
```

```
  118 NRP(I+2,2)=SIN(THAV)
      RETURN
C
C  SET ERROR FLAG ON
C
  999 IEF=1
C
C  RETURN WITH NAME ERROR
C
      RETURN
 1000 CONTINUE
C
C  EXAMINE PART NAME AND ASSIGN PART ID
C
      IF (NAME.EQ.'box') IDP=1
      IF (NAME.EQ.'cyl') IDP=2
C
C  VALID NAME?
C
      IF (IDP.EQ.9999) GO TO 999
C
C  SELECT SECTION OF CODE FROM PART ID
      GO TO (1010,2010),IDP
 1010 CONTINUE
      N1(1)=3
      N2(1)=3
      N3(1)=4
      N4(1)=4
C
C  VERTEX INFORMATION
C
      GO TO (1011,1012,1013,1014,1015,1016,1017,1018) NV
 1011 N1(2)=1
      N1(3)=4
      N1(4)=5
      N2(2)=2
      N2(3)=4
      N2(4)=5
      GO TO 1020
 1012 N1(2)=1
      N1(3)=2
      N1(4)=5
      N2(2)=1
      N2(3)=3
      N2(4)=6
      GO TO 1020
 1013 N1(2)=2
      N1(3)=3
      N1(4)=5
      N2(2)=2
      N2(3)=4
      N2(4)=7
      GO TO 1020
 1014 N1(2)=3
      N1(3)=4
      N1(4)=5
      N2(2)=1
      N2(3)=3
      N2(4)=8
      GO TO 1020
 1015 N1(2)=1
      N1(3)=4
      N1(4)=6
      N2(2)=1
      N2(3)=6
      N2(4)=8
      GO TO 1020
 1016 N1(2)=1
      N1(3)=2
      N1(4)=3
      N2(2)=2
      N2(3)=5
      N2(4)=7
      GO TO 1020
 1017 N1(2)=2
      N1(3)=3
      N1(4)=6
      N2(2)=3
      N2(3)=6
      N2(4)=8
      GO TO 1020
 1018 N1(2)=3
      N1(3)=4
      N1(4)=6
      N2(2)=4
      N2(3)=5
      N2(4)=7
 1020 CONTINUE
      GO TO (1021,1022,1023,1024,1025,1026) NF
 1021 N3(2)=2
      N3(3)=4
      N3(4)=5
      N3(5)=6
      N4(2)=1
      N4(3)=2
      N4(4)=6
      N4(5)=5
      GO TO 1030
 1022 N3(2)=1
      N3(3)=3
      N3(4)=5
      N3(5)=6
      N4(2)=2
      N4(3)=3
      N4(4)=7
      N4(5)=6
      GO TO 1030
 1023 N3(2)=2
      N3(3)=4
      N3(4)=5
      N3(5)=6
      N4(2)=3
      N4(3)=4
      N4(4)=8
      N4(5)=7
      GO TO 1030
 1024 N3(2)=1
      N3(3)=3
      N3(4)=5
      N3(5)=6
      N4(2)=4
      N4(3)=1
      N4(4)=5
      N4(5)=8
      GO TO 1030
 1025 N3(2)=1
      N3(3)=2
      N3(4)=3
      N3(5)=4
      N4(2)=1
      N4(3)=4
      N4(4)=3
      N4(5)=2
      GO TO 1030
 1026 N3(2)=1
      N3(3)=2
      N3(4)=3
      N3(5)=4
      N4(2)=5
      N4(3)=6
      N4(4)=7
      N4(5)=8
      GO TO 1030
 1030 CONTINUE
 1100 CONTINUE
      RETURN
 2010 CONTINUE
      N1(1)=3
      N2(1)=3
      N3(1)=4
      IF (NF.LE.2) N3(1)=NSEG
      N4(1)=4
      IF (NF.LE.2) N4(1)=NSEG
      N2(2)=NV+1
      IF (NV.EQ.NSEG) N2(2)=1
      IF (NV.EQ.(2*NSEG)) N2(2)=NSEG+1
      N2(3)=NV-1
      IF (NV.EQ.1) N2(3)=NSEG
      IF (NV.EQ.(NSEG+1)) N2(3)=2*NSEG
      N2(4)=NV+NSEG
      IF (NV.GT.NSEG) N2(4)=NV-NSEG
      IF (NV.LE.NSEG) N1(2)=1
      IF (NV.GT.NSEG) N1(2)=2
      N1(3)=2+NV
      IF (NV.GT.NSEG) N1(3)=2+NV-NSEG
      N1(4)=NV+3
      IF (NV.GT.NSEG) N1(4)=NV-NSEG+3
      IF (NV.EQ.NSEG) N1(4)=3
      IF (NV.EQ.(2*NSEG)) N1(4)=3
      IF (NF.EQ.1) NCH=1
      IF (NF.EQ.2) NCH=2
      IF (NF.GT.2) NCH=3
      GO TO (2011,2021,2031) NCH
 2011 DO 2014 I=1,NSEG
      N3(I+1)=I+2
 2014 CONTINUE
      GO TO 2041
 2021 DO 2024 I=1,NSEG
      N3(I+1)=I+2
 2024 CONTINUE
      GO TO 2041
 2031 N3(2)=1
      N3(3)=2
      N3(4)=NF+1
      IF (NF.EQ.(2+NSEG)) N3(4)=3
      N3(5)=NF-1
      IF (NF.EQ.3) N3(5)=2+NSEG
 2041 CONTINUE
      GO TO (2111,2121,2131) NCH
 2111 DO 2114 I=1,NSEG
      N4(1+I)=I
 2114 CONTINUE
      GO TO 2141
 2121 DO 2124 I=1,NSEG
      N4(1+I)=NSEG+I
 2124 CONTINUE
      GO TO 2141
 2131 NFV=NF-2
      N4(2)=NFV-1
      N4(3)=NFV
      N4(4)=NFV+NSEG
      N4(5)=NFV+NSEG-1
      IF (NFV.EQ.1) N4(2)=NSEG
      IF (NFV.EQ.1) N4(5)=2*NSEG
 2141 CONTINUE
      RETURN
      END
```

APPENDIX V.3- GENPRO Source Listing

```
        PROGRAM genpro
C
c       This program generates an output file containing the robot and
c       table positions from the files generated by AUTOWELD.
C
C*******************************************************************
C
c       File names.

        character*20 ptable,plocat,wpoint,wtable,corect
        character*20 itable,ilocat
        character*20 progra

        real values(10,3)

        integer recl(20)

c       Define sizes of file record lengths.
c       Files: cad,cadlst,wpcdat,procad,ptable,plocat,itable,ilocat,wpoint,
c              wtable,corect

        recl(1)=36
        recl(2)=8
        recl(3)=20
        recl(4)=52
        recl(5)=8
        recl(6)=40
        recl(7)=8
        recl(8)=40
        recl(9)=4
        recl(10)=16
        recl(11)=24

c       Define file names.

        ptable='PTABLE.DAT'
        plocat='PLOCAT.DAT'
        itable='ITABLE.DAT'
        ilocat='ILOCAT.DAT'
        wpoint='WPOINT.DAT'
        wtable='WTABLE.DAT'
        corect='CORECT.DAT'
        progra='PROGRA.DAT'


c          Start the program.

        open(unit=1,file=ptable,form='unformatted',access='direct',
     *      status='old',recl=recl(5))
        open(unit=2,file=plocat,form='unformatted',access='direct',
     *            status='old',recl=recl(6))
        open(unit=3,file=wpoint,form='unformatted',access='direct',
     *            status='old',recl=recl(9))
```

```
        open(unit=4,file=wtable,form='unformatted',access='direct',
     *            status='old',recl=recl(10))
        open(unit=5,file=itable,form='unformatted',access='direct',
     *            status='old',recl=recl(7))
        open(unit=6,file=ilocat,form='unformatted',access='direct',
     *            status='old',recl=recl(8))
        open(unit=7,file=progra,status='new')


        write(7,*) 'ROBOT WELDING PROGRAM GENERATED OFFLINE BY AUTOP'
        write(7,*)

        read(1,rec=1) nwelds

        do 10 iseam=1,nwelds

c          Initialize the next weld sequence.

           write(7,*)
           write(7,*)
           write(7,*) '********** WELD ',iseam,' **********'
           write(7,*)
           write(7,*) '***** Inter-weld positions *****'
           write(7,*)
           write(7,*) '       Robot joint positions
     *                 Table joint        Time'
           write(7,*) '
     *                   positions       intervals'
           write(7,*)
           read(5,rec=iseam) nstart,nend
           index=0
           do 20 ipos=nstart,nend
              index=index+1
              read(6,rec=ipos) (values(i,1),i=1,6),(values(i,2),i=1,3),time
              write(7,100) index,(values(i,1),i=1,6),(values(i,2),i=1,3),time
20         continue
100        format(i3.3x,6f6.1,2x,3f6.1,2x,f6.2)

c          Replay the weld.

           write(7,*)
           write(7,*)
           write(7,*) '***** Weld trajectory *****'
           write(7,*)
           write(7,*) '       Robot joint positions
     *                 Table joint        Time'
           write(7,*) '
     *                   positions       intervals'
           write(7,*)

           read(1,rec=iseam+1) istart,iend
           index=0
           do 30 ipos=istart,iend
              index=index+1
              read(2,rec=ipos) (values(i,1),i=1,6),
     *        (values(j,2),j=1,3),time
```

```
              write(7,100) index,(values(i,1),i=1,6),
     *        (values(j,2),j=1,3),time
30         continue

           write(7,*)
           write(7,*)
           write(7,*) '***** Welding parameters *****'
           write(7,*)
           write(7,*) 'voltage, wire feed rate, gas flow rate, current'
           write(7,*)

           index=0
           do 60 ipos=istart,iend
              index=index+1
              read(3,rec=ipos) ipoint
              read(4,rec=ipoint) volt,feed,gfr,wcurr
              write(7,200) index,volt,feed,gfr,wcurr
60         continue
200        format(i3,7x,4f8.2)
10      continue

        read(5,rec=nwelds+1) nstart,nend

        write(7,*)
        write(7,*)
        write(7,*) '***** Return to rest *****'
        write(7,*)
           write(7,*) '       Robot joint positions
     *                 Table joint         Time'
           write(7,*) '
     *                   positions       intervals'
        write(7,*)

        index=0
        do 40 ipos=nstart,nend
           index=index+1
           read(6,rec=ipos) (values(i,1),i=1,6),(values(i,2),i=1,3),time
           write(7,100) index,(values(i,1),i=1,6),(values(i,2),i=1,3),time
40      continue

        write(7,*)
        write(7,*)
        write(7,*) '********** END OF PROGRAM **********'
        write(7,*)

        stop
        end
```

APPENDIX V.4- Face Splitting Source Listings

```
c*******************************************************************************
        program main
c
c       This is the main interactive graphics program for
c       testing interference detection based
c       on hidden surface removal.
c
c       SUBROUTINES CALLED:
c               drint-  draws the volume of intersection
c               drmodl- draws the objects in the scene
c               hsurf-  finds a set of non-intersecting faces and orders
c                       them in a priority list for output
c               interf- finds the faces defining the intersection volume
c               rdmodl- reads the geometry data file.
c               sort-   does a preliminary depth sort of object faces
c
c       IRIS GRAPHICS CALLS:
c               clear,color,double,gconfi,gexit,gflush,ginit,loadma,
c               mapcol,polarv,popmat,pushma,qbutto,qkeybo,rotate,
c               setval,swapbu,transf,transl,window
c
c       VARIABLES:
c               forder(100)-    priority ordered list of faces
c               idend(4,4)-     4 X 4 identity matrix
c               intvol(100)-    priority ordered list of faces of intersection
c                               volume
c               part(100)-      list of parts to be processed
c               tpdist(100)-    face visibility (for debugging only)
c
c*******************************************************************************
c
C
$INCLUDE /usr/include/fdevice.h
$INCLUDE /usr/include/fgl.h
$include /usr/ralph/model/common
$include /usr/ralph/model/newell

$include /usr/ralph/model/debug

C
        REAL IDENT(4,4)
C
        INTEGER*2 Y,Z,ROTX,ROTY
        integer*2 icolor,imap

        integer part(100)
        integer pair(2)
        integer forder(100),intvol(100)

        real tpdist(100)

c
c       initialize global data
c
        reset=0
        obscur=1
```

```
      parent=2
      outsid=3
      splitp=4
      splitq=5
      spltpq=6
C
c     initialize an identity matrix
c
      DO 1 I=1,4
         IDENT(I,I)=1.
1     CONTINUE
c
c     set the forth element of all homogeneous vectors to 1.
c
      do 103 i=1,100
         vcoord(4,i)=1.
         fnorms(4,i)=1.
103   continue
C
c
c     initialize graphics
c
      CALL GINIT
C
      call color(blue)
      call clear


c     set up double buffer mode and map a grey scale.

      CALL DOUBLE
      call gconfi
      do 80 imap=0,15
         icolor=16*imap
      CALL MAPCOL(imap,icolor,icolor,icolor)
80    continue

      call swapbu
      do 81 imap=0,15
         icolor=16*imap
         call mapcol(imap,icolor,icolor,icolor)
81    continue


c     read the data base from the geometry file.

      CALL RDMODL


c     set up buttons and valuators

      CALL QKEYBO
      CALL SETVAL(DIAL0,I2(500),I2(0),I2(10000))
      CALL SETVAL(DIAL1,I2(0),I2(-1800),I2(1800))
      CALL SETVAL(DIAL2,I2(0),I2(-1800),I2(1800))
      CALL SETVAL(DIAL3,I2(0),I2(-1800),I2(1800))
```

```
        call SETVAL(DIAL4,i2(0),i2(-1000),i2(1000))
        call SETVAL(DIAL5,i2(0),i2(-1000),i2(1000))
        call SETVAL(DIAL6,i2(0),i2(-1000),i2(1000))
        call setval(dial7,i2(0),i2(-1000),i2(1000))
        call QBUTTO(mouse2)
C
c       UNTIL (qtest().ne.0)
c
10      CONTINUE

c       open a file for debugging output

        open(2,file='test.dat',status='new')
c
c       get valuator values and assign to variables

        X = FLOAT(GETVAL(DIAL0))/100.
        Y= GETVAL(DIAL1)
        Z= GETVAL(DIAL2)
        ROTX =GETVAL(DIAL3)
        ROTY=GETVAL(DIAL7)
        xtran=float(getval(dial4))/100.
        ytran=float(getval(dial5))/100.
        ztran=float(getval(dial6))/100.



c       perform viewing and modelling transformations on data base

        CALL PUSHMA
        CALL LOADMA(IDENT)
        call polarv(x,y,z,i2(0))
        ivert1=parts(3,1)
        ivert2=ivert1+parts(4,1)-1
        inorm1=parts(1,1)
        inorm2=inorm1+parts(2,1)-1
        call tran(ivert1,ivert2,inorm1,inorm2)
        ivert1=parts(3,2)
        ivert2=ivert1+parts(4,2)-1
        inorm1=parts(1,2)
        inorm2=inorm1+parts(2,2)-1
        call transl(xtran,ytran,ztran)
        CALL ROTATE(ROTX,'X')
        call rotate(roty,'y')
        call tran(ivert1,ivert2,inorm1,inorm2)
        CALL POPMAT

c       sort all faces by screen depth

        part(1)=1
        part(2)=2
        call sort(2,part)
        call hsurf(forder)
        pair(1)=1
        pair(2)=2
        call interf(pair,intvol,intot)
```

```
c       define a window and draw the transformed data base

        CALL window(-1.,1.,-1.,1.,2.,1000.)
        call color(blue)
        call clear
        call drmodl(forder,itotal)
        call drint(intvol,intot)


        call gflush
        CALL SWAPBU

C
        IF(QTEST().eq.0) then
           close(2)
           goto 10
        endif

c       END UNTIL

        CALL GEXIT


c       calculate face visibility
c
        do 101 i=1,fend
        ivert=verts(faces(1,i))
        do 102 j=1,3
           tpdist(i)=tpdist(i)+tcoord(j,ivert)*tnorms(j,i)
102     continue
        if (tpdist(i).gt.0) then
           tpdist(i)=0
        else
           tpdist(i)=1
        endif


c       output debugging data to file

101     continue
        write(2,*) 'fend ',fend,'vend ',vend,'vcend ',vcend
        write(2,*) 'itotal ',itotal
        do 99 i=1,fend
           write(2,*) ifsort(1,i),ifsort(2,i),ifsort(3,i)
99      continue
        write(2,*) 'forder '
        do 98 i=1,itotal
           write(2,*) forder(i),tpdist(forder(i))
98      continue
        write(2,*) 'intvol '
        do 888 i=1,intot
           write(2,*) i,intvol(i)
888     continue
        write(2,*) 'faces'
```

```
      do 95 i=1,fend
         write(2,*) i,(faces(j,i),j=1,2)
95    continue
      write(2,*) 'verts '
      do 96 i=1,vend
         write(2,*) i,verts(i)
96    continue
      write(2,*) 'tcoord '
      do 97 i=1,vcend
         write(2,*) i,(tcoord(j,i),j=1,3)
97    continue
      write(2,*) 'tnorms'
      do 999 i=1,fend
         write(2,*) i,(tnorms(j,i),j=1,3)
999   continue
      write(2,*) ' scoord '
      do 89 i=1,vcend
         write(2,*) i,(scoord(j,i),j=1,3)
89    continue
      do 100 i=1,fend
         write(2,*) (box(j,i),j=1,6)
100   continue
      write(*,*)


      CLOSE(2)

      STOP
      END
C


C***********************************************************************
      INTEGER FUNCTION I2(I)
C
C     THIS FUNCTION CONVERTS TYPE INTEGER TO TYPE INTEGER*2
c     This function is machine dependent.
C
c***********************************************************************
      INTEGER*2 M
      EQUIVALENCE (M,N)
      M=I
      I2=N
C
      RETURN
      END
```

```
        SUBROUTINE drint(ifsort,intot)
C
c       This subroutine draws the volume of intersection.
c       The volume is depicted as a wire frame.

c       CALLED FROM-
c               main

c       SUBROUTINES CALLED-
c               drpoly

c       IRIS GRAPHICS CALLS-
c               color, draw, move
c

c       PARAMETERS-
c               ifsort(100)-    the array of faces to be drawn
c               intot-          the number of faces

c*******************************************************************************
C
$INCLUDE /usr/include/fdevice.h
$INCLUDE /usr/include/fgl.h
$include /usr/ralph/model/common
C
        integer ifsort(100),itotal
        INTEGER ISTART,INDEX
        integer*2 shade
        real array3(3,3), array4(3,4), array5(3,5), array6(3,6), array7(3,7)

C
C
           DO 20 J=1,intot
              iface=ifsort(j)
              ISTART=VERTS(FACES(1,iface))
              call color(i2(0))
              CALL MOVE(tCOORD(1,ISTART),TCOORD(2,ISTART),TCOORD(3,ISTART))
              DO 30 K=FACES(1,iface),FACES(1,iface)+FACES(2,iface)-1
                 INDEX=VERTS(K)
                 CALL DRAW(TCOORD(1,INDEX),TCOORD(2,INDEX),TCOORD(3,INDEX))
30            CONTINUE
              CALL DRAW(TCOORD(1,ISTART),TCOORD(2,ISTART),TCOORD(3,ISTART))
20         CONTINUE
C
C
        RETURN
        END
        subroutine drpoly(numb,index,array)

c       This subroutine maps the face vertices into an array and then draws
c       a filled polygon.

c       CALLED FROM:    drmodl

c       IRIS GRAPHICS CALLS:
c               polf

c       PARAMETERS:
c               numb-   the number of vertices on the face
c               index-  the location of the first index in the vertex list
c               array-  the array of vertices
c*******************************************************************************
$include /usr/ralph/model/common

        integer numb,index
        real array(3,numb)

        iarray=1
              do 35 icount=index,index+numb-1
                 ivert=verts(icount)
                 do 36 kk=1,3
                    array(kk,iarray)=tcoord(kk,ivert)
36               continue
           iarray=iarray+1
35            continue
              call polf(numb,array)
        return
        end
        subroutine hsurf(forder)

c       hidden surface removal using the Newell/Newell/Sancha algorithm

c       CALLED FROM: main

c       SUBROUTINES CALLED:
c               htests
c               resort
c               sort2
c               split
c               swap

c       PARAMETERS:
c               forder- the resulting priority ordered list of faces

c       VARIABLES:
c               iface(2,2)-     array containing part no. and index of two face
c               ilist(10)-      a separate list containing split face fragments
c               pindex-         the index of the current head of the face list
c               qindex-         the upper bound index defining the set of
c                               canditate faces
c               findex-         the number of faces currently sorted
c
c*****************************************************************************/
c

$include /usr/include/fdevice.h
$include /usr/include/fgl.h
$include /usr/ralph/model/common
$include /usr/ralph/model/newell

$include /usr/ralph/model/debug

        integer forder(100)
        integer iface(2,2)
        logical stop,result,spflag,psplit
        integer pindex,qindex,findex
        integer ilist(10)

c       initialize parameters

        pindex=1
        findex=1
        fend=nface
        vend=nvl
        vcend=nvert
        fnew=0
        debug=.false.


c       repeat until all faces have been ordered

c       WHILE (findex.lt.itotal)

        ifsort(3,pindex)=reset
```

```
10      if (findex.ge.itotal) goto 20

c       perform a hierarchy of tests on the current face against all of the
c       other faces

           call htests(pindex,qindex,stop)


           if(.not.stop) then


c       test each possibly obscured face against the current face

              call sort2(pindex,qindex,result,ilist,iflag)
              if (iflag.le.0.or.iflag.ge.100) write(2,*) ' iflag error',iflag

c       swap the current face with an obscured face, or split the current
c       face by the plane of a mutually obscuring face

              if (result) then
                 call swap(pindex,iflag)
              else

              psplit=.false.
              do 55 ifac=1,iflag
                 if(.not.psplit.and.(ifsort(3,ilist(ifac)).eq.splitp.or.
     *              ifsort(3,ilist(ifac)).eq.spltpq)) then
                    do 56 i=1,2
                       iface(i,1)=ifsort(i,pindex)
                       iface(i,2)=ifsort(i,ilist(ifac))
56                  continue
                    call split(iface,spflag)
                    ifsort(3,pindex)=parent
                    psplit=.true.
                 endif

                 if(ifsort(3,ilist(ifac)).eq.splitq.or.ifsort(3,ilist(ifac))
     *           .eq.spltpq) then
                    do 57 i=1,2
                       iface(i,1)=ifsort(i,ilist(ifac))
                       iface(i,2)=ifsort(i,pindex)
57                  continue
                    call split(iface,spflag)
                    ifsort(3,ilist(ifac))=parent
                 endif
55            continue

                 call resort(pindex,2*(iflag+1))
                 fnew=0
              endif

c       insert the current face into the ordered face list and remove it
c       from the list of faces to be sorted

           else
              forder(findex)=ifsort(2,pindex)
              ifsort(3,pindex)=reset
              pindex=pindex+1
              findex=findex+1
           endif

        if (pindex.ge.100) write(2,*) ' pindex too large'
c       END WHILE

        goto 10
20      continue

c       the last remaining face goes at the end of the list

           forder(findex)=ifsort(2,pindex)

        itotal=findex

        return
        end
        subroutine htests(pindex,qindex,stop)

c       perform a heirarchy of hidden surface tests on a candidate face P
c       against a set of faces {Q} according to a modified Newell/Newell/Sancha
c       algorithm

c       CALLED FROM:
c               hsurf

c       SUBROUTINES CALLED:
c               ovrlap
c               plside

c       PARAMETERS:
c               pindex- the index of the current candidate face P in the list
c               qindex- the index of the last face of the set {q} in the list
c               stop-   .true. if all of the tests succeed and face P can be
c                       output to the ordered face list
c
c*******************************************************************************

$include /usr/ralph/model/common
$include /usr/ralph/model/newell

$include /usr/ralph/model/debug

        integer fpair(2)
        integer pxmin,pxmax,pymin,pymax,pzmin,pzmax
        integer pindex,qindex
        logical stop,result


        qindex=pindex
        ifsort(3,pindex)=reset


c       find the set of faces {Q} which overlap P in z

c       WHILE (box(5,ifsort(2,qindex)).lt.box(6,ifsort(2,pindex)))

        pzmin=box(5,ifsort(2,pindex))
        stop=.true.
20      if (box(6,ifsort(2,qindex+1)).lt.pzmin.or.qindex+1.gt.itotal) goto 10
           qindex=qindex+1
           if (ifsort(1,pindex).ne.ifsort(1,qindex)) then
              ifsort(3,qindex)=obscur
              stop=.false.
           else
              ifsort(3,qindex)=reset
           endif
c       END WHILE
        goto 20
10      continue
```

```
c       test bounding boxes of P and Q in x and y.

        if (.not.stop) then
        stop=.true.
        pxmax=box(2,ifsort(2,pindex))
        pxmin=box(1,ifsort(2,pindex))
        pymin=box(3,ifsort(2,pindex))
        pymax=box(4,ifsort(2,pindex))
        do 30 index=pindex+1,qindex
           if (ifsort(3,index).eq.obscur) then
              if (box(1,ifsort(2,index)).gt.pxmax.or.
     *            box(2,ifsort(2,index)).lt.pxmin.or.
     *            box(3,ifsort(2,index)).gt.pymax.or.
     *            box(4,ifsort(2,index)).lt.pymin) then
              ifsort(3,index)=reset

              else
                 stop=.false.
              endif
           endif
30      continue
        endif


c       test if P is wholly on far side of Q and if Q is wholly on near
c       side of P.


        if (.not.stop) then
           stop=.true.
           fpair(1)=ifsort(2,pindex)
           do 50 index=pindex+1,qindex
              if (ifsort(3,index).eq.obscur) then
                 fpair(2)=ifsort(2,index)
                 call plside(fpair,result,isplit)
                 if (result) then
                    ifsort(3,index)=reset
                 else
                    stop=.false.
                    ifsort(3,index)=isplit
                 endif
              endif
50         continue
        endif

c       test if the projections of P and Q are disjoint

        if (.not.stop) then
           stop=.true.
           fpair(1)=ifsort(2,pindex)
           result=.false.
           do 60 index=pindex+1,qindex
              if (ifsort(3,index).ne.reset) then
                 fpair(2)=ifsort(2,index)
                 call ovrlap(fpair,result)
                 if (.not.result) then
```

```
                    ifsort(3,index)=reset
                 else
                    stop=.false.
                 endif
              endif
60         continue
        endif

        return
        end
        subroutine interf(part,intvol,icount)

c       This subroutine finds the set of faces which define the boundary
c       of the volume of intersection.  First, all of the faces which
c       are interior to all of the origional faces of one of the convex
c       parts.  Then, all of the faces which are interior to all of the
c       faces just found are determined.  The total set of faces defines
c       the boundary of the volume of intersection.

c       CALLED FROM-
c               main

c       SUBROUTINES CALLED-
C               none

c       PARAMETERS-
c               part(2)- the part numbers of the two intersecting parts
c               intvol(100)-    an array of faces defining the intersection
c               icount-         the number of faces in 'intvol'

c*******************************************************************************

$include /usr/ralph/model/common
$include /usr/ralph/model/newell

        integer part(2)
        integer intvol(100)
        logical inside,onface

        toler=.01
        icount=0

c       test each part against the other

        do 5 ipart=1,2
        iother=ipart+1
        if (ipart.eq.2) iother=1
        istart=parts(1,part(ipart))
        iend=istart+parts(2,part(ipart))-1

c       test each face fragment of one part against each face of the
c       other part. A face fragment is inside a part (and is a face of the
c       volume of intersection) if it is inside every face of of the part.

        do 10 iface1=istart,iend

c       find the distance of the first face plane to the origin

           dist1=0.
           do 15 i=1,3
              dist1=dist1+tcoord(i,verts(faces(1,iface1)))*tnorms(i,iface1)
15         continue

           do 20 iface2=1,fend
              if (ifsort(1,iface2).eq.part(iother).and.
     *        ifsort(3,iface2).eq.reset) then
```

```
                 ivert=faces(1,ifsort(2,iface2))
                 ivend=ivert+faces(2,ifsort(2,iface2))-1
                 inside=.true.
                 onface=.true.

                 do 25 ivrt=ivert,ivend
                 if (inside.and.onface) then

c       a face fragment is outside of the test face if any of its vertices
c       gives a greater distance to the origin when plugged into the plane
c       equation.

                 dist2=0.
                 do 30 i=1,3
                    dist2=dist2+tcoord(i,verts(ivrt))*tnorms(i,iface1)
30               continue
                 if (dist2.gt.(dist1+toler)) then
                    inside=.false.
                    onface=.false.
                    ifsort(3,iface2)=outsid
                 else
                    if (dist2.lt.(dist1-toler)) then
                       onface=.false.
                    endif
                 endif
                 endif
25               continue

c       test a face fragment which lies on the test face to see if its
c       surface normal is the same or opposite to the test face.
c       If it is opposite, the fragment is outside.

                 if (onface) then
                    tdot=0.
                    do 26 i=1,3
                       tdot=tdot+tnorms(i,iface1)*tnorms(i,iface2)
26                  continue
                    if (tdot.lt.0.) ifsort(3,iface2)=outsid
                 endif

              endif
20         continue
10      continue

c       store the faces of the intersection volume in a separate array

        do 40 ilist=1,fend
           if (ifsort(1,ilist).eq.part(iother).and.
     *     ifsort(3,ilist).eq.reset) then
              icount=icount+1
              intvol(icount)=ifsort(2,ilist)
           endif
40      continue
5       continue

        return
```

```
      end
```

```
        subroutine ovrlap(iface,result)

c       test if the projections of two polygonal faces overlap on the screen

c       CALLED FROM:
c               htests

c       SUBROUTINES CALLED:
C               none

c       PARAMETERS:
c               iface(2)-       indices of two faces two be compared
c               result-         .true. if the faces overlap in screen space
c*******************************************************************************

$include /usr/ralph/model/common
$include /usr/ralph/model/newell

$include /usr/ralph/model/debug

        integer iface(2),istart(2),inumb(2)
        integer iend(2)
        logical result

        toler=1.5


c       find the face vertices in the vertex list

        do 10 i=1,2
           istart(i)=faces(1,iface(i))
           inumb(i)=faces(2,iface(i))
           iend(i)=istart(i)+inumb(i)-1
10      continue


c       the polygon screen projections are disjoint if all vertices of
c       one face lie "outside" of any edge of the other face.

        result=.true.

c       test each face against the other

        do 15 ipoly=1,2
        if (result) then
           do 20 iverts=istart(ipoly),iend(ipoly)
              if (result) then
                 icurr=verts(iverts)
                 inext=verts(iverts+1)
                 if (iverts.eq.iend(ipoly)) inext=verts(istart(ipoly))

c       find the line equation coefficients of an edge

                 dx=scoord(1,inext)-scoord(1,icurr)
                 dy=scoord(2,inext)-scoord(2,icurr)
                 dlen=sqrt(dx**2+dy**2)
                 if (dlen.ne.0.) then
                    dx=dx/dlen
                    dy=dy/dlen
                    dist=-dy*scoord(1,icurr)+dx*scoord(2,icurr)

c       find which side of the edge faces "outward"

                    test1=0.
                    do 88 ivrt2=istart(ipoly),iend(ipoly)
                       if (abs(test1).le..0001) then
                          itest=verts(ivrt2)
                          test1=-dy*scoord(1,itest)+dx*scoord(2,itest)-dist
                       endif
88                  continue

c       plug all the vertices of the other face into the line equation.
c       If they all lie "outside" the edge, the faces are disjoint.

                    iother=2
                    if(ipoly.eq.2) iother=1
                    result=.false.
                    do 25 ivert2=istart(iother),iend(iother)
                       if (.not.result) then
                          icurr2=verts(ivert2)
                          test2=-dy*scoord(1,icurr2)+dx*scoord(2,icurr2)-dist
                          if(test1*test2.gt.0.and.abs(test2).gt.toler)
     *                       result=.true.
                       endif
25                  continue
                 endif
              endif
20         continue
           endif
15      continue

22      return
        end
        subroutine plside(iface,result,isplit)

c       test if face(1) is wholly on far side of plane of face(2), or if face(2)
c       is wholly on near side of plane of face(1)

c       CALLED FROM:
c               htests

c       SUBROUTINES CALLED:
c               none

c       PARAMETERS:
C               iface(2)-       indices of the two faces to be compared
c               result-         .true. if neither face is split by the plane of
c                               the other face
c               isplit-         if result is .false., returns as one of
c                               the following values:
c                                  splitp-  split face(1)
c                                  splitq-  split face(2)
c                                  splitpq- split both faces
c*******************************************************************************

$include /usr/ralph/model/common
$include /usr/ralph/model/newell

$include /usr/ralph/model/debug

        integer iface(2),istart(2),inumb(2),vstart(2)
        real dist(2),norm(3,2)
        logical result,flag1,flag2

        toler=.0001
        result=.true.
        isplit=0


c       find the face vertices in the vertex list

        do 10 i=1,2
           istart(i)=faces(1,iface(i))
           inumb(i)=faces(2,iface(i))
           vstart(i)=verts(istart(i))
10      continue


c       calculate the distances of the planes from the origin

        do 15 i=1,2
           dist(i)=0.
           do 16 j=1,3
              norm(j,i)=tnorms(j,iface(i))
              dist(i)=dist(i)+tcoord(j,vstart(i))*norm(j,i)
16         continue
15      continue
```

```
c       test if face 1 is wholly on far side of plane of face 2

        flag1=.true.
           do 20 iverts=istart(1),istart(1)+inumb(1)-1
              if (flag1) then
                 icurr=verts(iverts)
                 dotprd=0.
                 do 25 i=1,3
                    dotprd=dotprd+tcoord(i,icurr)*norm(i,2)
25               continue
                 if (dist(2).ge.0.) then
                    if (dotprd.lt.dist(2)-toler) flag1=.false.
                 else
                    if (dotprd.gt.dist(2)+toler) flag1=.false.
                 endif
                 if (.not.flag1) isplit=splitp
              endif
20         continue


c       test if face 2 is wholly on near side of plane of face 1

              flag2=.true.
              do 30 iverts=istart(2),istart(2)+inumb(2)-1
                 if (flag2) then
                    icurr=verts(iverts)
                    dotprd=0.
                    do 35 i=1,3
                       dotprd=dotprd+tcoord(i,icurr)*norm(i,1)
35                  continue
                    if (dist(1).ge.0.) then
                       if (dotprd.gt.dist(1)+toler) flag2=.false.
                    else
                       if (dotprd.lt.dist(1)-toler) flag2=.false.
                    endif
                    if (.not.flag2) then
                       if (isplit.eq.0) then
                          isplit=splitq
                       else
                          isplit=spltpq
                       endif
                    endif
                 endif
30            continue
              result=flag1.and.flag2
        return
        end
        SUBROUTINE RDMODL
C
C       THIS PROGRAM READS A POLYHEDRAL MODEL FROM A FILE AND
C       INITIALIZES THE DATA STRUCTURES in common block common.
C
C*******************************************************************************
$include /usr/ralph/model/common
C
C
C
C       READ THE GEOMETRY FILE
C
        OPEN(1,FILE='GEOM.DAT')
C
        READ(1,*) NPT
        DO 10 I=1,NPT
           READ(1,*) (PARTS(J,I),J=1,4)
10      CONTINUE
C
        READ(1,*) NFACE
        DO 20 I=1,NFACE
           READ(1,*) (FACES(J,I),J=1,2)
20      CONTINUE
C
        READ(1,*) NVL
        DO 30 I=1,NVL
           READ(1,*) VERTS(I)
30      CONTINUE
C
        READ(1,*) NVERT
        DO 40 I=1,NVERT
           READ(1,*) (VCOORD(J,I),J=1,3)
           VCOORD(4,I)=1.
40      CONTINUE
C
        READ(1,*) NFACE2
        DO 50 I=1,NFACE2
           READ(1,*) FACES2(I)
50      CONTINUE
C
        READ(1,*) NVADJ
        DO 60 I=1,NVADJ
           READ(1,*) AJVERT(I)
60      CONTINUE
C
        READ(1,*) NFNORM
        DO 70 I=1,NFNORM
           READ(1,*) (FNORMS(J,I),J=1,3)
           FNORMS(4,I)=1.
70      CONTINUE
C
        do 80 i=1,nfnorm
           do 85 j=1,3
              fnorms(j,i)=fnorms(j,i)+vcoord(j,verts(faces(1,i)))
85         continue
```

```
80      continue

        CLOSE(1)
C
        RETURN
        END
        subroutine resort(pindex,num)

c       find the bounding boxes of the new face segments, and re-sort the
c       face list, inserting the new face segments in the correct places

c       CALLED FROM:
c               hsurf

c       SUBROUTINES CALLED:
c               none

c       PARAMETERS:
c               pindex- index pointing to the current head of the unsorted face
c                       list
c               num-    the number of new faces to be inserted into the
c                       face list
c
c********************************************************************************

$include /usr/ralph/model/common
$include /usr/ralph/model/newell

        integer x,y,z,xmin,xmax,ymin,ymax,zmin,zmax
        integer*2 postra(3)
        integer inew(10)
        integer iftemp(3)
        integer pindex
        real pretra(3)
        logical istat,insert


c       find the bounding boxes

        index=1
        do 10 iface=fend-num+1,fend
           inew(index)=iface
           index=index+1
           ivstrt=faces(1,iface)
           ivend=ivstrt+faces(2,iface)-1
           xmin=1000
           xmax=-xmin
           ymin=xmin
           ymax=xmax
           zmin=xmin
           zmax=xmax
           do 20 ivert=ivstrt,ivend
              do 25 i=1,3
                 pretra(i)=tcoord(i,verts(ivert))
25         continue
           call screen(1,pretra,postra,istat)
           x=postra(1)
           if(x.lt.xmin) xmin=x
           if(x.gt.xmax) xmax=x

           y=postra(2)
           if(y.lt.ymin) ymin=y
```

```
            if(y.gt.ymax) ymax=y

            z=postra(3)
            if(z.lt.zmin) zmin=z
            if(z.gt.zmax) zmax=z
20          continue

            box(1,iface)=xmin
            box(2,iface)=xmax
            box(3,iface)=ymin
            box(4,iface)=ymax
            box(5,iface)=zmin
            box(6,iface)=zmax
10       continue


c        remove parent faces from face list

         imax=itotal
         iface=pindex
         do 100 icount=pindex,imax
            if (ifsort(3,iface).eq.parent) then
               itotal=itotal-1
               do 110 iface2=iface,itotal
                  do 120 i=1,3
                     ifsort(i,iface2)=ifsort(i,iface2+1)
120               continue
110            continue
            else
               iface=iface+1
            endif
100      continue

c        remove the parent faces from the new face  list

         imax=fnew
         iface=1
         do 101 icount=1,imax
            if (ifnew(3,iface).eq.parent) then
               fnew=fnew-1
               do 111 iface2=iface,fnew
                  do 121 i=1,3
                     ifnew(i,iface2)=ifnew(i,iface2+1)
121               continue
111            continue
            else
               iface=iface+1
            endif
101      continue

c        sort the face fragments

         do 150 i=1,fnew-1
            do 160 j=i+1,fnew
               if (box(6,ifnew(2,j)).gt.box(6,ifnew(2,i))) then
                  do 170 k=1,3
```

```
                    iftemp(k)=ifnew(k,j)
                    ifnew(k,j)=ifnew(k,i)
                    ifnew(k,i)=iftemp(k)
170              continue
              endif
160        continue
150     continue


c       insert the face fragments into the face list



        do 200 ind1=1,fnew
           ilimit=itotal
           insert=.true.
           do 210 ind2=pindex,ilimit
              if (insert) then
              if (box(6,ifsort(2,ind2)).lt.box(6,ifnew(2,ind1))) then
                 do 220 ind3=itotal,ind2,-1
                    do 230 kk=1,3
                       ifsort(kk,ind3+1)=ifsort(kk,ind3)
230                 continue
220              continue

                 itotal=itotal+1

                 do 240 kk=1,3
                    ifsort(kk,ind2)=ifnew(kk,ind1)
240              continue

                 insert=.false.

              endif
              endif
210        continue
           if (insert) then
              itotal=itotal+1
              do 250 ii=1,3
                 ifsort(ii,itotal)=ifnew(ii,ind1)
250           continue
           endif
200     continue

        return
        end
```

```
      subroutine screen(n,pretra,postra,istat)

c     This subroutine emulates the IRIS command 'screen' which doesn't work.
c     Screen transforms an array of world coordinate vectors into screen
c     coordinates including perspective depth into the screen.
c
c     CALLED FROM: sort

c     SUBROUTINES CALLED:     none

c     PARAMETERS:
c             n-      dimension of arrays
c             pretra- array of real world coordinate vectors
c             postra- array of vectors transformed into integer screen
c                     coordinates
c             istat-  a dummy logical array to match the IRIS command.
c*******************************************************************************


      integer n
      integer*2 postra(3,n)
      real pretra(3,n)
      logical istat(n)

      real temp1(4),temp2(4)

      do 10 i=1,n
         postra(1,n)=int(-2000.*pretra(1,n)/pretra(3,n))
         postra(2,n)=int(-2000.*pretra(2,n)/pretra(3,n))
         postra(3,n)=int(1000.*(1002.+4000./pretra(3,n))/998.)
10    continue

      return
      end
```

```
        subroutine sort(nparts,part)

c       This subroutine does a preliminary sort of the face list according to
c       maximum screen depth, and calculates the bounding boxes for each face.

c       CALLED FROM:
c               main
c
c       SUBROUTINES CALLED:
c               screen
c
c       PARAMETERS:
c               nparts-         the number of parts
c               part-           the list of parts

c       VARIABLES:
c               pretra(3)- a world coordinate vector
c               postra(3)- a transformed screen coordinate vector
c
c
c*******************************************************************************
c

$include /usr/ralph/model/common
$include /usr/ralph/model/newell

        integer part(100)
        integer x,y,z,xmin,xmax,ymin,ymax,zmin,zmax
        integer*2 postra(3)
        real pretra(3)
        integer iftemp(3)
        logical ivalid
        integer vstart


c       calculate the bounding boxes for all the faces

        index=0
        do 10 ipart=1,nparts
           itemp=parts(1,part(ipart))
           do 11 iface=itemp,itemp+parts(2,part(ipart))-1
              index=index+1
              vstart=faces(1,iface)
              xmin=10000
              xmax=-xmin
              ymin=xmin
              ymax=-xmin
              zmin=xmin
              zmax=-xmin
              do 15 ivert=vstart,vstart+faces(2,part(ipart))-1
                 icoord=verts(ivert)
                 do 50 i=1,3
                    pretra(i)=tcoord(i,icoord)
50               continue
                 call screen(1,pretra,postra,ivalid)
                 do 51 i=1,3
```

```
                  scoord(i,icoord)=postra(i)
51             continue
               x=postra(1)
               if (x.lt.xmin) xmin=x
               if (x.gt.xmax) xmax=x

               y=postra(2)
               if (y.lt.ymin) ymin=y
               if (y.gt.ymax) ymax=y

               z=postra(3)
               if (z.lt.zmin) zmin=z
               if (z.gt.zmax) zmax=z

15          continue
            box(1,index)=xmin
            box(2,index)=xmax
            box(3,index)=ymin
            box(4,index)=ymax
            box(5,index)=zmin
            box(6,index)=zmax

            ifsort(1,index)=ipart
            ifsort(2,index)=iface

11       continue
10    continue
      itotal=index


c     sort the face list by depth

      do 20 i=1,index-1
         do 25 j=i+1,index
            if (box(6,ifsort(2,j)).gt.box(6,ifsort(2,i))) then
               do 30 k=1,3
                  iftemp(k)=ifsort(k,j)
                  ifsort(k,j)=ifsort(k,i)
                  ifsort(k,i)=iftemp(k)
30             continue
            endif
25       continue
20    continue

      return
      end
```

```
        subroutine sort2(pindex,qindex,result,ilist,iflag)

c       This subroutine test pairs of possibly intersecting faces.
c       All faces in the face list which possibly intersect the current
c       face P are tested against face P.

c       CALLED BY-
C               hsurf

c       SUBROUTINES CALLED-
c               plside

c       PARAMETERS-
c               pindex- index of face P
c               qindex- the index of the last potentially intersecting face
c               result- .true. if there are no intersecting faces found
c               ilist-  list of faces found to intersect
c               iflag-  index of first potentially intersecting face if no
c                       faces are found to intersect

c********************************************************************************

$include /usr/ralph/model/common
$include /usr/ralph/model/newell

        integer pindex,qindex,iflag
        integer fpair(2)
        integer ilist(10)
        logical result,test

        result=.true.
        iflag=0
        fpair(2)=ifsort(2,pindex)

c       Test all of the faces not eliminated by the previous hierarchy of
c       tests.  All faces whose planes split face P or which are split by
c       the plane of face P are stored in the list 'ilist'.

        icount=0
        do 10 index=pindex+1,qindex
           if (ifsort(3,index).ne.reset) then
              if (iflag.eq.0) iflag=index
                 fpair(1)=ifsort(2,index)
                 call plside(fpair,test,isplit)

c       face intersection determined

                 if (.not.test) then
                    icount=icount+1
                    ilist(icount)=index
                    itest=ifsort(3,index)

c       face Q is split by plane of face P

                    if((itest.eq.splitq.and.isplit.eq.spltpq).or.
     *                 (itest.eq.spltpq.and.isplit.eq.splitp)) then
```

```
                     ifsort(3,index)=splitq
                  else

c       face P is split by plane of face Q

                  if((itest.eq.splitp.and.isplit.eq.spltpq).or.
     *               (itest.eq.spltpq.and.isplit.eq.splitq)) then
                     ifsort(3,index)=splitp
                  else

c       both faces are split

                  if(itest.eq.spltpq.and.isplit.eq.spltpq) then
                     ifsort(3,index)=spltpq
                  endif
                  endif
                  endif

                  result=.false.
                  iflag=icount

               endif
         endif
10      continue

        return
        end
        subroutine split(face,result)

c       This subroutine splits face P along the line of intersection
c       with the plane
c       of face Q.

c       CALLED FROM:
c               hsurf

c       SUBROUTINES CALLED:
c               screen

c       PARAMETERS:
c               face(2,2)-      array containing indexes and part numbers of
c                               two faces to be split
c               result-         logical variable returning true if a valid
c                               split was made

c       VARIABLES:
c               vl(3),v2(3)-    ends of a test edge
c               vint(3,2)-      array containing the two edge intersection
c                               points between one polygon and the plane of
c                               the other
c               varray(500)-    array containing face vertex list with
c                               intersection points inserted
c               fnorm(3,2)-     the surface normals of the two face planes
c               dist(2)-        the distances of the two planes from the origin
c
c*******************************************************************************
c

$include /usr/ralph/model/common
$include /usr/ralph/model/newell


        logical result
        real vl(3),v2(3),vint(3,2)
        real fnorm(3,2),dist(2)
        real pretra(3)
        integer*2 postra(3)
        logical istat
        integer face(2,2)
        integer ivert(2)
        integer flag(2)
        integer varray(500)


        result=.true.

        do 1 i=1,2
           if (face(2,i).le.0.or.face(2,i).gt.fend) then
           write(2,*) ' split-face error face(',i,')  ',face(2,i)
           write(2,*) ' pindex, iflag ',pindex,iflag
           endif
```

```
1     continue


c     find the surface normals and distances to the origin for each of the
c     two faces

      do 10 i=1,2
         dist(i)=0.
         ivert(i)=verts(faces(1,face(2,i)))
         do 15 j=1,3
            fnorm(j,i)=tnorms(j,face(2,i))
            dist(i)=dist(i)+tcoord(j,ivert(i))*fnorm(j,i)
15       continue
10    continue


c     find the intersection points where the boundary of face(1) intersects
c     the plane of face(2) and insert them
c     into a revised vertex list

      istart=faces(1,face(2,1))
      iend=istart+faces(2,face(2,1))-1
      numb=0
      do 20 icount=istart,iend
         varray(icount+numb)=verts(icount)
         if (numb.lt.2) then
         do 25 i=1,3
            v1(i)=tcoord(i,verts(icount))
            if (icount.lt.iend) then
               v2(i)=tcoord(i,verts(icount+1))
            else
               v2(i)=tcoord(i,verts(istart))
            endif
25       continue

         dot1=0
         dot2=0
         do 30 i=1,3
            dot1=dot1+v1(i)*fnorm(i,2)
            dot2=dot2+v2(i)*fnorm(i,2)
30       continue

         temp2=dot2-dot1
         if (abs(temp2).gt..00001) then
            tval=(dist(2)-dot1)/temp2
         else
            tval=1000.
         endif
         if (tval.ge.0..and.tval.lt.1.) then
            numb=numb+1
            flag(numb)=icount+numb
            do 35 i=1,3
               vint(i,numb)=v1(i)+(v2(i)-v1(i))*tval
35          continue

     if (abs(temp2).lt..01) then
```

```
         tdist=0.
         do 95 i=1,3
            tdist=tdist+vint(i,numb)*fnorm(i,2)
95    continue
         tdist=tdist-dist(2)
         write(2,*) ' tdist ' ,tdist
         endif

            vcend=vcend+1
            varray(icount+numb)=vcend
            do 40 i=1,3
               tcoord(i,vcend)=vint(i,numb)
               pretra(i)=vint(i,numb)
40          continue
            call screen(1,pretra,postra,istat)
            do 41 i=1,3
               scoord(i,vcend)=postra(i)
41          continue
         endif
         endif
20    continue

      if (numb.ne.2) then
         result=.false.
         vcend=vcend-numb
      endif

      if (numb.ne.0.and.numb.ne.2) write(2,*) '  split numb ',numb


c     separate the revised vertex list into two separate lists
c     for each face fragment and update FACES and VERTS.

      if (numb.eq.2) then
      icount=0
      do 50 i=istart,flag(1)
         icount=icount+1
         verts(vend+icount)=varray(i)

         if (verts(vend+icount).le.0.or.verts(vend+icount).gt.500)
     * write(2,*) 'split error- vend,icount,varray(i) ',vend,icount,varray(i)

50    continue
      do 55 i=flag(2),iend+numb
         icount=icount+1
         verts(vend+icount)=varray(i)

         if (verts(vend+icount).le.0.or.verts(vend+icount).gt.500)
     * write(2,*) 'split error- vend,icount,varray(i) ',vend,icount,varray(i)

55    continue


      fend=fend+1
      fnew=fnew+1
      ifnew(1,fnew)=face(1,1)
      ifnew(2,fnew)=fend
      ifnew(3,fnew)=reset
      faces(1,fend)=vend+1
      faces(2,fend)=icount
      do 58 i=1,3
         tnorms(i,fend)=fnorm(i,1)
58    continue
      vend=vend+icount

      icount=0
      do 51 i=flag(1),flag(2)
         icount=icount+1
         verts(vend+icount)=varray(i)
51    continue

      fend=fend+1
      fnew=fnew+1
      ifnew(1,fnew)=face(1,1)
      ifnew(2,fnew)=fend
      ifnew(3,fnew)=reset
      faces(1,fend)=vend+1
      faces(2,fend)=icount
      do 59 i=1,3
         tnorms(i,fend)=fnorm(i,1)
59    continue
      vend=vend+icount

      if (fend.ge.100.or.vend.ge.500) write(2,*) 'split- fend or vend error'

      endif

      return
      end
```

```
        subroutine swap(pindex,iflag)

c       This subroutine inserts the current face P into the face list according
c       to its maximum screen depth, and replaces it with the face whose index
c       is 'iflag'.

c       CALLED FROM-
c               hsurf

c       SUBROUTINES CALLED-
c               none

c       PARAMETERS-
c               pindex- index of the current head of the face list, which
c                       contains the current face P.
c               iflag-  the index of the face which will be moved to the
c                       head of the face list.

c*********************************************************************************

$include /usr/ralph/model/newell

        integer pindex, temp(3)
        logical insert

c       save the current face P and replace it in the face list by face Q

        if (pindex.ne.iflag) then
        do 10 i=1,3
           temp(i)=ifsort(i,pindex)
           ifsort(i,pindex)=ifsort(i,iflag)
10      continue

c       remove face Q from its previous location in the face list

        do 20 index=iflag,itotal-1
           do 30 i=1,3
              ifsort(i,index)=ifsort(i,index+1)
30         continue
20      continue

c       insert face P in the correct place in the face list according to
c       maximum screen depth

        insert=.false.
        do 40 icount=pindex+1,itotal-1
           if (.not.insert) then
              if(box(6,temp(2)).ge.box(6,ifsort(2,icount))) then
                 do 41 i=itotal-1,icount,-1
                    do 42 j=1,3
                       ifsort(j,i+1)=ifsort(j,i)
42                  continue
41               continue
                 do 45 i=1,3
                    ifsort(i,icount)=temp(i)
45               continue
```

```
                 insert=.true.
              endif
           endif
40      continue

c       if face P hasn't been inserted yet, then it goes at the end of the lis

        if (.not.insert) then
           do 50 i=1,3
              ifsort(i,itotal)=temp(i)
50         continue
        endif
        endif

        return
        end
        subroutine tran(ivert1,ivert2,inorm1,inorm2)

c       Transform the vertex coords in the vertex list
c       from index ivert1 to index ivert2
c       and transform the surface normals from index inorm1 to index inorm2
c       by the current transformation.
c
c       CALLED FROM:
c               main
c
c       SUBROUTINES CALLED:
c               none
c
c       IRIS GRAPHICS CALLS:
c               transf
c
c       PARAMETERS:
C               ivert1- starting vertex
c               ivert2- ending vertex
c               inorm1- starting normal
c               inorm2- ending normal

c       VARIABLES:
c               postra(4)- homogeneous vector after transformation
c               pretra(4)-      ''          ''   before      ''
c
c       COMMON VARIABLES:
c               fnorms(4,n)-    untransformed face normals
c               tnorms(4,n)-    transformed face normals
c               vcoord(4,n)-    untransformed vertex coordinates
c               tcoord(4,n)-    transformed vertex coordinates
c
c****************************************************************************

c

$include /usr/ralph/model/common

        real pretra(4),postra(4)

c       transform the range of vertices
c
        do 35 i=ivert1,ivert2
           do 38 j=1,4
              pretra(j)=vcoord(j,i)
38         continue
           call transf(1,pretra,postra)
           do 36 j=1,4
              tcoord(j,i)=postra(j)
36         continue
35      continue

c
c       transform the range of normals
c
        do 40 i=inorm1,inorm2
```

```
          do 41 j=1,4
             pretra(j)=fnorms(j,i)
41        continue
          call transf(1,pretra,postra)
          do 37 j=1,4
             tnorms(j,i)=postra(j)-tcoord(j,verts(faces(1,i)))
37        continue
40     continue
       return
       end
```

APPENDIX V.5- Duncan Methods Source Listings

```
C
C  THIS PROGRAM IS USED TO VIEW A ROBOT LINKAGE
C
C  This program was based on one written by Zhentao Zhang. Several
C  of the subroutines, as noted, were written by him and accepted
C  by me as correct.  Basically this program constructs a box
C  around both the object, in this case a finial and the arm,
C  in case a parellipiped.  The boxes are tested by a routine
C  called TB to see if they overlap if they do not then the
C  program moves on to the next position.  If the boxes do
C  overlap then each point in the finial is tested by TF to see if it
C  is inside the parallipiped.
C
C                          Alan Steeves, April 10, 1985
C
C  Transfered to the IRIS and the TEK IGL calls modified for
C  use with the IRIS Graphics Library.
C
C                          Alan Steeves, May 29, 1985
C
c  This program was further debugged and modified to work properly on
c  the IRIS.

c                          Ralph Buchal, March 4, 1986

C
C         SUBROUTINES REFERENCED:
C
C
C         AXES    A routine used to draw a 3D axes.
C
C         CINTER  A routine that checks to see if the two
C  object interfer with one another.
C
C         DRAW    An IRIS routine to draw in 3D.
C
C         DRWARM  A routine to draw the arm in 3D.
C
C         GEXIT   An IRIS routine to stop the graphic processing.
C
C         GINIT   An IRIS routine to start the graphic processing .
C
C         MOVE    An IRIS routine to do a 3D move.
C
C         PLTCLD  A routine that is used to plot the cloud
C  of interference points.
C
C         ROT     A routine to rotate an object.
C
C         ROTATE  An IRIS routine used to do a 3D rotation of the
C  viewing area.
C
C         SCALE   An IRIS routine used to scale the point to be
C  viewed.
C
C         VIEWPO  An IRIS routine used to set the 3D view port.
```

```
C
C          WINDOW  An IRIS routine used to specify the dimensions of
C     the 3D window.
C
C
C     This common block is used to pass parameters to some of the
C     subroutines.  It is used because the original program used a number of
C     common blocks and changing them all was getting to complicated.
C
      COMMON /COOR/ XFINAL, YFINAL, ZFINAL,
     1                XPATH, YPATH, ZPATH,
     2                XCLOUD, YCLOUD, ZCLOUD,
     3                XOVLAP, YOVLAP, ZOVLAP,
     4                XARM,YARM,ZARM
C
C          DEFINE THE VARIABLES USED
C
C     ANSWER contain the yes no answer for an questions that are
C     asked.
C
      CHARACTER*1 ANSWER
C
C     The following variables are used for colors
C
      INTEGER*2 BLKCOL, REDCOL, GRNCOL, YELCOL, BLUCOL
C
C     The next three logical variables are used to check if search
C     should be terminated after the first hit and to check if the
C     the cloud of interference points are to be plotted.
C
      LOGICAL*1 PLTINT, CHCKAL, GRPHON
C
C     XFINAL, YFINAL and ZFINAL contain the x, y and z points
C     for the finial.
C
      REAL XFINAL(29,36), YFINAL(29,36), ZFINAL(29,36)
C
C     XARM, YARM, and ZARM contain the points for the original
C     position of the arm.
C
      REAL XARM(8), YARM(8), ZARM(8)
C
C     XPOLD, YPOLD, and ZPOLD contains the postion of
C     linkage element that was just drawn.  This postion is used to
C     erase the old postion after the new one is drawn.
C
      REAL XPOLD(8), YPOLD(8), ZPOLD(8)
C
C     XPATH, YPATH and ZPATH contain the postion of the link element
C     that is currently being worked on.
C
      REAL XPATH(8), YPATH(8), ZPATH(8)
C
C     XCLOUD, YCLOUD and ZCLOUD contain the points in the finial
C     that are inside the link element.
C
```

```
   REAL XCLOUD(100), YCLOUD(100), ZCLOUD(100)
C
C  XOVLAP, YOVLAP and ZOVLAP contain the points on the
C  the finial that are in the common box.
C
   REAL XOVLAP(500), YOVLAP(500), ZOVLAP(500)
   REAL XAXISX(6), XAXISY(6), XAXISZ(6), YAXISX(6),
     1    YAXISY(6), YAXISZ(6), ZAXISX(6), ZAXISY(6),
     2    ZAXISZ(6)
   DATA XAXISX/0.0,45.0,45.0,47.0,45.0,45.0/
   DATA XAXISY/0.0, 0.0, 1.0, 0.0,-1.0, 0.0/
   DATA XAXISZ/0.0, 0.0, 0.0, 0.0, 0.0, 0.0/
   DATA YAXISX/0.0, 0.0, 1.0, 0.0,-1.0, 0.0/
   DATA YAXISY/0.0,45.0,45.0,47.0,45.0,45.0/
   DATA YAXISZ/0.0, 0.0, 0.0, 0.0, 0.0, 0.0/
   DATA ZAXISX/0.0, 0.0,-1.0, 0.0, 1.0, 0.0/
   DATA ZAXISY/0.0, 0.0, 0.0, 0.0, 0.0, 0.0/
   DATA ZAXISZ/0.0,45.0,45.0,47.0,45.0,45.0/
C
C  Set up the colors
C
   DATA BLKCOL, REDCOL, GRNCOL, YELCOL, BLUCOL/0, 1, 2, 3, 4/
C
C  MF is the number of segments around the finial, in this
C  case we have the finial divided into 36 segments of 10 degrees
C  each.  NF is the number of levels in the finial, in this case we
C  have 29 levels.
C
   MF = 36
   NF = 29
C
C  MA is the number of points around the arm, NA is the
C  number of levels in the arm.
C
   MA = 4
   NA = 2
C
C  READ THE DATA FOR THE FINIAL
C
   OPEN (1, FILE = 'FINIAL.COR')
   DO 100 J = 1, MF
          DO 200 I = 1, NF
          READ(1, 900, END = 999) XFINAL(I,J),
     1                                    YFINAL(I,J),
     2                                    ZFINAL(I,J)
200                CONTINUE
100       CONTINUE
900       FORMAT(3F20.6)
C
C  READ THE DATA FOR THE ARM
C
   OPEN (2, FILE = 'ARM.COR')
   DO 101 I = 1, MA*NA
          READ(2, *) XARM(I), YARM(I), ZARM(I)
101       CONTINUE
C
```

```
C  READ THE INITIALIZATION DATA
C  The file "robot.dat" contains information of the scale of the drawing
C  axes rotations etc.
C
   OPEN (3, FILE = 'ROBOT.DAT')
C
C  FIND THE SCALE
C
   read(3,*) xscale
C
C  FIND THE STEP SIZE
C  By making the step size larger we can reduce the number
C  of points that are plotted for the finial.  This will save
C  plotting time but result in a finial that is less well defined
C  and may result in a definition were the arm can be inside the
C  finial but not touching an of the points.
C
   READ(3, *) ISTEP
C
C  FIND THE ROTATIONS
C
   READ(3,*) IXROT, IYROT, IZROT
C
C  FIND THE STARTING POINT OF THE ARM
C  This allows us to move the starting point to any postion.
C
   READ(3, *) XARMST, YARMST, ZARMST
C
C  FIND THE DISPLACEMENT FOR THE FINIAL
C  This allows us to displace the finial to any postion.
C
   READ(3, *) XDISP, YDISP, ZDISP
C
C  FIND IF THE GRAPHICS ARE TO BE USED.
C
   WRITE(*, 9204)
9204        FORMAT(' Do you want graphics (Y/N) ? ',$)
   READ(*, 9211) ANSWER
   GRPHON = .TRUE.
   IF (ANSWER .EQ. 'N' .OR. ANSWER .EQ. 'n')
     1                            GRPHON = .FALSE.
C
C  IF NO GRAPHICS THEN SKIP THE NEXT QUESTION
C
   IF (GRPHON) GOTO 4567
   PLTINT = .FALSE.
   GOTO 5678
C
C  FIND IF THE INTERFERENCE POINTS ARE TO BE PLOTTED.
C
4567        WRITE(*, 9210)
9210        FORMAT(' Do you want the cloud of interference points to
     1 be plotted (Y/N) ? ',$)
   READ(*, 9211) ANSWER
9211        FORMAT(A1)
   PLTINT = .TRUE.
```

```
      IF (ANSWER .EQ. 'N' .OR. ANSWER .EQ. 'n')
     1                              PLTINT = .FALSE.
C
C  FIND IF THE SEARCH SHOULD BE STOPPED AFTER THE FIRST HIT
C
5678        WRITE(*, 9215)
9215        FORMAT(' Do you want to stop the search after the first
     1 hit (Y/N) ? ',$)
      READ(*, 9211) ANSWER
      CHCKAL = .TRUE.
      IF (ANSWER .EQ. 'Y' .OR. ANSWER .EQ. 'y')
     1                              CHCKAL = .FALSE.
C
C  IF THE GRAPHICS ARE OFF SKIP THE INITIALIZATION
C
      IF (.NOT. GRPHON) GOTO 9911
C
C  Initialize the graphics.
C
      CALL GINIT
      CALL COLOR(BLUCOL)
      CALL CLEAR
      CALL DOUBLE
      CALL GCONFI
      CALL MAPCOL(I2(1),I2(0),I2(255),I2(0))
      CALL MAPCOL(I2(3),I2(255),I2(255),I2(0))
      CALL MAPCOL(I2(4),I2(0),I2(0),I2(255))
      CALL SWAPBU
      CALL MAPCOL(I2(1),I2(255),I2(0),I2(0))
      CALL MAPCOL(I2(3),I2(255),I2(255),I2(0))
      CALL MAPCOL(I2(4),I2(0),I2(0),I2(255))
C
C  The next call sets up the dimensions of the 3D window.
C
      CALL WINDOW(-50.0, 150.0, -70.0, 130., 2000., 20000.)
      call polarv(3000.,i2(0),i2(0),i2(0))

C
C  Rotate the axis by the amount entered previously.
C
      CALL ROTATE(I2(IXROT),'X')
      CALL ROTATE(I2(IYROT),'Y')
      CALL ROTATE(I2(IZROT),'Z')
C
C  Scale the screen by the amount entered.
C
      CALL SCALE(XSCALE, XSCALE, XSCALE)
C
C  MAKE AN OBJECT FOR FOR THE AXIS
C
      axes=genobj()
      CALL MAKEOB(AXES)
C
C  X-AXES
C
      CALL MOVE(XAXISX(1), XAXISY(1), XAXISZ(1))
```

```
   DO 1 I= 2,6
           CALL DRAW (XAXISX(I), XAXISY(I), XAXISZ(I))
1  CONTINUE
C
C  CREATE A LABEL X FOR THE X-AXIS
C
   CALL MOVE(48.0,1.5,0.0)
   CALL DRAW(50.0,-1.5,0.0)
   CALL MOVE(50.0,1.5,0.0)
   CALL DRAW(48.0,-1.5,0.0)
C
C  Y-AXIS
C
   CALL MOVE(YAXISX(1), YAXISY(1), YAXISZ(1))
   DO 2 I= 2,6
           CALL DRAW (YAXISX(I), YAXISY(I), YAXISZ(I))
2  CONTINUE
C
C  CREATE A LABEL Y FOR THE Y AXIS
C
   CALL MOVE(-1.5,50.0,0.0)
   CALL DRAW(0.0,48.0,0.0)
   CALL MOVE(1.5,50.0,0.0)
   CALL DRAW(-1.5,47.0,0.0)
C
C  Z AXIS
C
   CALL MOVE (ZAXISX(1), ZAXISY(1), ZAXISZ(1))
   DO 3 I= 2,6
           CALL DRAW (ZAXISX(I), ZAXISY(I), ZAXISZ(I))
3  CONTINUE
C
C  CREATE A LABEL Z FOR THE Z-AXIS
C
   CALL MOVE(-1.5,1.5,47.5)
   CALL DRAW(1.5,1.5,47.5)
   CALL DRAW(-1.5,-1.5,47.5)
   CALL DRAW(1.5,-1.5,47.5)
C
C  CLOSE THE OBJECT
C
   CALL CLOSEO(AXES)
C
C  MAKE AN OBJECT OF THE FINIAL
C
   finial=genobj()
   CALL MAKEOB(FINIAL)
9911       DO 300 J = 1, MF, ISTEP
   XFINAL(1,J) = XFINAL(1,J) + XDISP
   YFINAL(1,J) = YFINAL(1,J) + YDISP
   ZFINAL(1,J) = ZFINAL(1,J) + ZDISP
   IF (.NOT. GRPHON) GOTO 9912
   CALL MOVE( XFINAL(1, J), YFINAL(1,J), ZFINAL(1,J))
9912       CONTINUE
           DO 400 I = 2, NF
           XFINAL(I,J) = XFINAL(I,J) + XDISP
           YFINAL(I,J) = YFINAL(I,J) + YDISP
           ZFINAL(I,J) = ZFINAL(I,J) + ZDISP
           IF (.NOT. GRPHON) GOTO 400
           CALL DRAW( XFINAL(I,J), YFINAL(I,J), ZFINAL(I,J))
400                CONTINUE
300        CONTINUE
   IF (.NOT. GRPHON) GOTO 9913
C
C  DRAW TOP AND BOTTOM RINGS
C  This code draws the circles on the top and bottom.
C
   CALL MOVE( XFINAL(1, 1), YFINAL(1,1), ZFINAL(1,1))
   DO 301 J = 1, MF, ISTEP
           CALL DRAW( XFINAL(1,J), YFINAL(1,J), ZFINAL(1,J))
301        CONTINUE
           CALL DRAW( XFINAL(1, 1), YFINAL(1,1), ZFINAL(1,1))
C
C
   CALL MOVE( XFINAL(29, 1), YFINAL(29,1), ZFINAL(29,1))
   DO 302 J = 1, MF, ISTEP
           CALL DRAW(XFINAL(29,J),YFINAL(29,J),ZFINAL(29,J))
302        CONTINUE
           CALL DRAW(XFINAL(29,1), YFINAL(29,1),ZFINAL(29,1))
C
C  CLOSE THE OBJECT
C
   CALL CLOSEO(FINIAL)
C
C  Start drawing the graphics
C
   CALL COLOR(BLUCOL)
   CALL CLEAR
   CALL COLOR(YELCOL)
   CALL CALLOB(AXES)
   CALL CALLOB(FINIAL)
9913       CONTINUE
C
C  CALCULATE AND PLOT THE ARM PATH
C
   DO 550 IJ = 1, MA*NA
           XPATH(IJ) = XARM(IJ) + XARMST
           YPATH(IJ) = YARM(IJ) + YARMST
           ZPATH(IJ) = ZARM(IJ) + ZARMST
550                CONTINUE
C
C  CHECK TO SEE IF WE SHOUD DRAW
C
   IF (.NOT. GRPHON) GOTO 9914
   CALL DRWARM( MA*NA, XPATH, YPATH, ZPATH )
C
C  SWAP THE BACK BUUFER TO THE FRONT TO TURN ON THE PICTURE
C
   CALL SWAPBU
9914       CONTINUE
C
C  CHECK FOR INTERFERENCE
```

```
C  The variable INTER contains the number of interference points
C  that were found.
C
   INTER = 0
   CALL CINTER (MF, NF, ISTEP, MA, NA, INTER,CHCKAL)
C
C  If any interference points found then plot them
C
   IF ( INTER .GT. 0 .AND. GRPHON ) CALL PLTCLD
     1          (XCLOUD, YCLOUD, ZCLOUD, INTER, PLTINT)
   IF ( INTER .GT. 0 .AND. (.NOT. GRPHON) )
     1          WRITE(*, 950)
950         FORMAT(' INTERFERENCE FOUND AT T = 0 ')
C
C
   XMRROT = 0.
   YMRROT = 0.
   ZMRROT = 0.
C
C  The 500 loop displaces the arm and checks for interference.
C  NOTE: this loop displaces every point of the arm by it's x, y
C        and z arm_start, then adds a function of (it) to the x value,
C             a function of (it*it) to the y value and a function of
C             (it*it*it ) to the z value.  The value (it) can be thought of
C             as being time and we are calculating the arm postion at various
C             times.  In a real case these postions would be calculated by
C             another program and read from a file by this one.
C
   DO 500 IT = 1, 20
           DO 600 IJ = 1, MA*NA
           XPATH(IJ) = XARM(IJ)  + FLOAT(IT) + XARMST
           YPATH(IJ) = YARM(IJ)  + (FLOAT( IT * IT ) / 4.)
     1                                        +  YARMST
           ZPATH(IJ) = ZARM(IJ)  + (FLOAT( IT * IT * IT) / 80.)
     1                                        +  ZARMST
600                 CONTINUE
C
C  ROTATE THE ARM
C  Here the arm rotations are again a function of (it). NOTE: that the
C  number of degrees that the arm rotates increases with increasing (it).
C
   XMRROT = XMRROT  +  FLOAT(IT)
   YMRROT = YMRROT  +  FLOAT(IT)
   ZMRROT = ZMRROT  +  FLOAT(IT)
   CALL ROT(  MA*NA, XMRROT, YMRROT, YMRROT)
C
C  CHECK TO SEE IF WE SHOUD DRAW
C
   IF (.NOT. GRPHON) GOTO 9915
C
C  DRAW THE ARM
C
   CALL COLOR(BLUCOL)
   CALL CLEAR
   CALL COLOR(YELCOL)
   call gflush
```

```
      CALL CALLOB(AXES)
      CALL CALLOB(FINIAL)
      CALL DRWARM(MA*NA, XPATH, YPATH, ZPATH )
9915        CONTINUE
C
C  CHECK FOR INTERERENCE
C
      INTER = 0
      CALL CINTER (MF, NF, ISTEP, MA, NA, INTER,CHCKAL)
C
C  PLOT CLOUD
C
      IF ( INTER .GT. 0 .AND. GRPHON ) CALL PLTCLD
     1         (XCLOUD, YCLOUD, ZCLOUD, INTER, PLTINT)
C
      IF ( INTER .GT. 0 .AND. (.NOT. GRPHON) )
     1         WRITE(*, 951) IT
951         FORMAT(' INTERFERENCE FOUND AT T =', I3)
C
      call swapbu
C  LOOP BACK FOR MORE POSTIONS
C
500         CONTINUE
C
C  END THE GRAPHICS
C
C
C  CHECK TO SEE IF WE SHOUD DRAW
C
      IF (.NOT. GRPHON) GOTO 998
      CALL GEXIT
C
C  FINISHED
C
C
C  STOP THE PROGRAM
C
998         STOP
C
C  Error messages
C
999         WRITE(*,1234) I*J
1234        FORMAT(' END OF FILE AT ',I8)
      STOP
      END
C
C  SUBROUTINES
C
C
      SUBROUTINE ROT( N, XROT, YROT, ZROT )
C
C  THIS SUBROUTINE WILL ROTATE THE POINTS DESCRIBED
C  BY THE X , Y , AND Z ARRAYS. X IS THE DEPTH AWAY
C  FROM THE USER Z IS THE VERITCAL COMPONENT.
C
      COMMON /COOR/ XFINAL, YFINAL, ZFINAL,
     1                XPATH, YPATH, ZPATH,
     2                XCLOUD, YCLOUD, ZCLOUD,
     3                XOVLAP, YOVLAP, ZOVLAP,
     4                XARM,YARM,ZARM
      REAL TEMP(8)
      REAL XFINAL(29,36), YFINAL(29,36), ZFINAL(29,36)
      REAL XARM(8), YARM(8), ZARM(8)
      REAL XPATH(8), YPATH(8), ZPATH(8)
      REAL XCLOUD(100), YCLOUD(100), ZCLOUD(100)
      REAL XOVLAP(500), YOVLAP(500), ZOVLAP(500)
            IF( N .LE. 0 ) RETURN
            IF( N .GT. 8 ) RETURN
C *** CALCULATE THE SINES AND COSINES .
            RC= .01745329
            T = XROT * RC
            SXROT = SIN( T )
            CXROT = COS( T )
            T = YROT * RC
            SYROT = SIN( T )
            CYROT = COS( T )
            T = ZROT * RC
            SZROT = SIN( T )
            CZROT = COS( T )
C **** CALCULATE CONSTANTS
            XC1 = CYROT*CZROT
            XC2 = SZROT*CYROT
            XC3 = SYROT
            YC1 = SXROT*SYROT*CZROT - CXROT*SZROT
            YC2 = SXROT*SYROT*SZROT + CXROT*CZROT
            YC3 = SXROT*CYROT
            ZC1 = CXROT*SYROT*CZROT + SXROT*SZROT
            ZC2 = CXROT*SYROT*SZROT - SXROT*CZROT
            ZC3 = CXROT*CYROT
C ***** NOW ROTATE IT.
            DO 100 I = 1 , N
            XT = XPATH(I)
            YT = YPATH(I)
            ZT = ZPATH(I)
            XPATH(I) = XT*XC1 + YT*XC2 - ZT*XC3
            YPATH(I) = XT*YC1 + YT*YC2 + ZT*YC3
            ZPATH(I) = XT*ZC1 + YT*ZC2 + ZT*ZC3
 100         CONTINUE
            RETURN
            END
C
C  SUBROUTINE TO DRAW THE ARM
C  This routine just does the draws and moves necessary to draw
C  the box that we are using for the arm.  In a real example this routine
C  would have to replaced with one that draws the arm as defined by
C  the user.
C
      SUBROUTINE DRWARM(N, X, Y, Z)
      REAL X(N), Y(N), Z(N)
      CALL MOVE( X(1), Y(1), Z(1))
      CALL DRAW( X(3), Y(3), Z(3))
      CALL DRAW( X(4), Y(4), Z(4))
      CALL DRAW( X(2), Y(2), Z(2))
      CALL DRAW( X(1), Y(1), Z(1))
      CALL DRAW( X(7), Y(7), Z(7))
      CALL DRAW( X(5), Y(5), Z(5))
      CALL DRAW( X(6), Y(6), Z(6))
      CALL DRAW( X(8), Y(8), Z(8))
      CALL DRAW( X(7), Y(7), Z(7))
      CALL MOVE( X(8), Y(8), Z(8))
      CALL DRAW( X(2), Y(2), Z(2))
      CALL MOVE( X(4), Y(4), Z(4))
      CALL DRAW( X(6), Y(6), Z(6))
      CALL MOVE( X(5), Y(5), Z(5))
      CALL DRAW( X(3), Y(3), Z(3))
      RETURN
      END
C
C
C
C
      SUBROUTINE TB(MF, NF, MA, NA,CONTER)
C
C     THIS IS A SUBROUTINE DO TEST WHETHER TWO BOX ARE OVERLAPED
C     NOTES: This routine was written by Zhentao Zhang.  It was modified
C     slightly by me to make it fit the format of this program but the
C     body of the routine is how he wrote it.
C
      COMMON /COOR/ XFINAL, YFINAL, ZFINAL,
     1              XPATH, YPATH, ZPATH,
     2              XCLOUD, YCLOUD, ZCLOUD,
     3              XOVLAP, YOVLAP, ZOVLAP,
     4              XARM,YARM,ZARM
      INTEGER T,CONTER
      LOGICAL CDTX,CDTY,CDTZ
      REAL XFINAL(29,36), YFINAL(29,36), ZFINAL(29,36)
      REAL XARM(8), YARM(8), ZARM(8)
      REAL XPATH(8), YPATH(8), ZPATH(8)
      REAL XCLOUD(100), YCLOUD(100), ZCLOUD(100)
      REAL XOVLAP(500), YOVLAP(500), ZOVLAP(500)
      REAL X3(8),Y3(8),Z3(8)
      REAL X4(8),Y4(8),Z4(8)
      REAL X6(16),Y6(16),Z6(16)
      REAL X8(16),Y8(16),Z8(16)
      REAL U(11,11,11),V(11,11,11),W(11,11,11)
C
      MNF=MF*NF
      MNA=MA*NA
C
C     FIND THE BOX FOR THE ROBOT ARM
C
      CALL BOX(XPATH,MNA,XS1,XL1)
      CALL BOX(YPATH,MNA,YS1,YL1)
      CALL BOX(ZPATH,MNA,ZS1,ZL1)
C
C     FIND THE BOX FOR THE FINIAL
C
      CALL BOX(XFINAL,MNF,XS,XL)
      CALL BOX(YFINAL,MNF,YS,YL)
      CALL BOX(ZFINAL,MNF,ZS,ZL)
C
C  DIVIDE THE FINIAL BOX SURFACE INTO MESH
C
88    CALL MESH(U,V,W,XS,YS,ZS,XL,YL,ZL)
C
C  DECIDE WHETHER BOX ARE OVERLAPED
C
      CONTER=0
      DO 10 I=1,11
      DO 10 J=1,11
      DO 10 K=1,11
      XO=U(I,J,K)
      YO=V(I,J,K)
      ZO=W(I,J,K)
      CDTX=((XO.GE.XS1).AND.(XO.LE.XL1))
      CDTY=((YO.GE.YS1).AND.(YO.LE.YL1))
      CDTZ=((ZO.GE.ZS1).AND.(ZO.LE.ZL1))
      IF((CDTX.AND.CDTY).AND.CDTZ) CONTER=CONTER+1
      IF((CDTX.AND.CDTY).AND.CDTZ) GO TO 79
      GO TO 10
79    XOVLAP(CONTER)=XO
      YOVLAP(CONTER)=YO
      ZOVLAP(CONTER)=ZO
10            CONTINUE
17    RETURN
      END
C
C
      SUBROUTINE TF(MF, NF, ISTEP, V0, X22, Y22, Z22,
     1       CONTER, INTER, CHCKAL)

C
C  THIS IS A SUBROUTINE DO TEST WHETHER ROBOT ARM INVADE FINIAL
C  NOTES: This routine was written by Zhentao Zhang.  It was modified
C  slightly by me to make it fit the format of this program but the
C  body of the routine is how he wrote it.
C                                               Alan Steeves 9 May 1985.
C
      COMMON /COOR/ XFINAL, YFINAL, ZFINAL,
     1              XPATH, YPATH, ZPATH,
     2              XCLOUD, YCLOUD, ZCLOUD,
     3              XOVLAP, YOVLAP, ZOVLAP,
     4              XARM,YARM,ZARM
      INTEGER T,CONTER,INTER
      LOGICAL*1 CND1,CND2,CND3,CD1,CD2,CD3, CHCKAL
      CHARACTER*26 NOI,INTF
              DIMENSION X22(4,2), Y22(4,2), Z22(4,2)
      REAL XA,XB,XC,YA,YB,YC,ZA,ZB,ZC
              REAL XD,YD,ZD,XO,YO,ZO,VOL
      REAL XFINAL(29,36), YFINAL(29,36), ZFINAL(29,36)
      REAL XARM(8), YARM(8), ZARM(8)
      REAL XPATH(8), YPATH(8), ZPATH(8)
      REAL XCLOUD(100), YCLOUD(100), ZCLOUD(100)
      REAL XOVLAP(500), YOVLAP(500), ZOVLAP(500)
```

```
           REAL*8 VV
C
   MNF=MF*NF
   VV = 0.D0
C
C  FIND THE SIZE OF THE COMMON BOX
C
   CALL BOX(XOVLAP,CONTER,XS2,XL2)
   CALL BOX(YOVLAP,CONTER,YS2,YL2)
   CALL BOX(ZOVLAP,CONTER,ZS2,ZL2)
C
C  CHOOSE & TEST INTERFERED POINTS
C
   INTER=0
   DO 11 K = 1, NF
           DO 10 I = 1, MF, ISTEP
           CND1 = ((XFINAL(K,I) .GE. XS2)
     1                   .AND. (XFINAL(K,I) .LE. XL2))
           CND2 =((YFINAL(K,I) .GE. YS2)
     1                   .AND. (YFINAL(K,I) .LE. YL2))
           CND3 = ((ZFINAL(K,I) .GE. ZS2)
     1                   .AND. (ZFINAL(K,I).LE.ZL2))
           IF((CND1.OR.CND2).OR.CND3) GO TO 2
           GO TO 10
2          XO=XFINAL(K,I)
           YO=YFINAL(K,I)
           ZO=ZFINAL(K,I)
C
           VV = 0.D0
           CALL VLM(XO,YO,ZO,X22,Y22,Z22,VV)
C
           XX = ABS(VV - V0)
           IF(ABS(VV-V0) .GT.0.1) GOTO 1011
C
C  FOUND INTERFERENCE
C
           IF (CHCKAL) GOTO 999
           WRITE (*, 900) 7
900        FORMAT(' ',A1,' INTERFERENCE FOUND, PROGRAM STOPPED. ')
           STOP
C
C
999                        INTER = INTER + 1
                   XCLOUD(INTER) = XO
                   YCLOUD(INTER) = YO
                   ZCLOUD(INTER) = ZO
1011               VV=0.D0
10                 CONTINUE
11 CONTINUE
17 RETURN
   END
C
C
   SUBROUTINE VLM(XO,YO,ZO,X,Y,Z,VV)
C
C  THIS IS A PROGRAM TO FIND THE VOLUME
```

```
C  NOTES: This routine was written by Zhentao Zhang.  It was not modified
C  in any way and is used as it was written.
C                                                Alan Steeves 9 May 1985.
C
   INTEGER T
         DIMENSION X(4,2), Y(4,2), Z(4,2), X5(5,2), Y5(5,2), Z5(5,2)
c  REAL*8 XA,XB,XC,YA,YB,YC,ZA,ZB,ZC,XD,YD,ZD,XO,YO,ZO
c  REAL*8 AX,BX,CX,AY,BY,CY,AZ,BZ,CZ,DX,DY,DZ,OX,OY,OZ
         REAL*8 VV,V
C
   DO 11 J=1,2
   X5(1,J)=X(1,J)
   X5(5,J)=X(1,J)
   X5(2,J)=X(2,J)
   X5(3,J)=X(3,J)
   X5(4,J)=X(4,J)
   Y5(1,J)=Y(1,J)
   Y5(5,J)=Y(1,J)
   Y5(2,J)=Y(2,J)
   Y5(3,J)=Y(3,J)
   Y5(4,J)=Y(4,J)
   Z5(1,J)=Z(1,J)
   Z5(5,J)=Z(1,J)
   Z5(2,J)=Z(2,J)
   Z5(3,J)=Z(3,J)
   Z5(4,J)=Z(4,J)
11 CONTINUE
   UX=(X(1,2)+X(3,2))/2
   VX=(X(1,1)+X(3,1))/2
   UY=(Y(1,2)+Y(3,2))/2
   VY=(Y(1,1)+Y(3,1))/2
   UZ=(Z(1,2)+Z(3,2))/2
   VZ=(Z(1,1)+Z(3,1))/2
C
C  CALCULATE THE VOLUNE
C
   N1=1
   DO 100 II=1,4
   DO 100 JJ=1,1
C
C  OPERATE ON TRIANGLE ABC AND ABD
C
   XA=X5(II,JJ)
   YA=Y5(II,JJ)
   ZA=Z5(II,JJ)
   XB=X5(II+1,JJ+1)
   YB=Y5(II+1,JJ+1)
   ZB=Z5(II+1,JJ+1)
   XC=X5(II+1,JJ)
   YC=Y5(II+1,JJ)
   ZC=Z5(II+1,JJ)
   BX=XB
   BY=YB
   BZ=ZB
   CX=XC
   CY=YC
```

```
      CZ=ZC
      XD=X5 (II,JJ+1)
      YD=Y5 (II,JJ+1)
      ZD=Z5 (II,JJ+1)
      DX=XD
      DY=YD
      DZ=ZD
      T=0
C
C   FIND THE VOLUME OF TETRAHEDRON OABC
C
66    DET1=XA*YC*ZO+XC*YO*ZA+XO*YA*ZC-XO*YC*ZA-XC*YA*ZO-XA*YO*ZC
      DET2=XA*YB*ZC+XB*YC*ZA+XC*YA*ZB-XC*YB*ZA-XB*YA*ZC-XA*YC*ZB
      DET3=- (XB*YC*ZO+XC*YO*ZB+XO*YB*ZC-XO*YC*ZB-XC*YB*ZO-XB*YO*ZC)
      DET4=- (XA*YB*ZO+XB*YO*ZA+XO*YA*ZB-XO*YB*ZA-XA*YO*ZB-XB*YA*ZO)
      V= (DET1+DET2+DET3+DET4) /6.
      V=ABS (V)
C
C   FIND ACCUMULATED VOLUME OF ALL TETRAHEDRONS
C
      VV=VV+V
      IF  (T.EQ.1)  GO TO 555
      IF  (T.EQ.2)  GO TO 666
      IF  (T.EQ.3)  GO TO 99
C
C   EXCHANGE C FOR D TO CALCULATE TRIANGLE ABD
C
      XC=XD
      YC=YD
      ZC=ZD
      T=1
      GO TO 66
C
C   DEAL WITH THE BOTTOM PARTS (EXCHANGE B FOR Q1(THE CENTER OF DOWN SIDE))
C
555           IF(JJ.NE.1) GO TO 666
      XB=VX
      YB=VY
      ZB=VZ
      XC=CX
      YC=CY
      ZC=CZ
      T=2
      GO TO 66
C
C   DEAL WITH TOP PARTS (EXCHANG A FOR Q(THE CENTER OF THE UP SIDE))
C
666           IF(JJ.NE.N1) GO TO 99
      XA=UX
      YA=UY
      ZA=UZ
      XB=BX
      YB=BY
      ZB=BZ
      XC=DX
      YC=DY
      ZC=DZ
      T=3
      GO TO 66
 99   T=0
 100          CONTINUE
      RETURN
      END
C
C
           SUBROUTINE BOX(P,N,PMIN,PMAX)
C
C     THIS SUBROUTINE CAN FIND THE MIN & MAX OF COORDINATES
C     NOTES: This routine was written by Zhentao Zhang.  It was not modified
C     in any way and is used as it was written.
C                                                  Alan Steeves 9 May 1985.
C
      REAL P(1)
C
      PMIN=P(1)
      PMAX=P(1)
      DO 1 I=1,N
      IF(P(I)-PMAX) 3,3,2
 2    PMAX=P(I)
 3    IF(P(I)-PMIN) 4,1,1
 4    PMIN=P(I)
 1    CONTINUE
      RETURN
      END
C
C
      SUBROUTINE MESH(U,V,W,XS,YS,ZS,XL,YL,ZL)
C
C     THIS IS A SUBROUTINE TO FIND MID POINTS OF A BOX
C     NOTES: This routine was written by Zhentao Zhang.  It was not modified
C     in any way and is used as it was written.
C                                                  Alan Steeves 9 May 1985.
C
      REAL U(11,11,11),V(11,11,11),W(11,11,11)
C
      DEX=(XL-XS)/10.
      DEY=(YL-YS)/10.
      DEZ=(ZL-ZS)/10.
      DO 1 I=1,11
      DO 1 J=1,11
      DO 1 K=1,11
      U(I,J,K)=XS+(I-1)*DEX
      V(I,J,K)=YS+(J-1)*DEY
      W(I,J,K)=ZS+(K-1)*DEZ
 1    CONTINUE
      RETURN
      END
C
C
      SUBROUTINE CH12(X1,Y1,Z1,X2,Y2,Z2)
C
C     THIS SUBROUTINE TRANSFORM A 1-D ARRAY TO A 2-D ONE
```

```
C  NOTES: This routine was written by Zhentao Zhang.  It was not modified
C  in any way and is used as it was written.
C                                                  Alan Steeves 9 May 1985.
C
   REAL X1(8),Y1(8),Z1(8),X2(4,2),Y2(4,2),Z2(4,2)
C
   DO 1 I=1,4
   X2(I,1)=X1(2*I-1)
   X2(I,2)=X1(2*I)
 1 CONTINUE
   DO 2 I=1,4
   Y2(I,1)=Y1(2*I-1)
   Y2(I,2)=Y1(2*I)
 2 CONTINUE
   DO 3 I=1,4
   Z2(I,1)=Z1(2*I-1)
   Z2(I,2)=Z1(2*I)
 3 CONTINUE
   RETURN
   END
C
C  THIS SUBROUTINE CHECKS FOR INTERFERENCE
C  Basically it just makes repeated calls to the routines
C  written by Zhentao Zhang.  In fact it is just a piece of code that
C  was taken from his original program and turned into a subroutine.
C                                                  Alan Steeves 9 May 1985.
C
   SUBROUTINE CINTER (MF,NF,ISTEP,MA,NA,INTER,CHCKAL)
   COMMON /COOR/ XFINAL, YFINAL, ZFINAL,
     1               XPATH, YPATH, ZPATH,
     2               XCLOUD, YCLOUD, ZCLOUD,
     3               XOVLAP, YOVLAP, ZOVLAP,
     4               XARM,YARM,ZARM
   INTEGER CONTER
   LOGICAL*1 CHCKAL
   REAL X2(4,2), Y2(4,2), Z2(4,2)
   REAL XFINAL(29,36), YFINAL(29,36), ZFINAL(29,36)
   REAL XARM(8), YARM(8), ZARM(8)
   REAL XPATH(8), YPATH(8), ZPATH(8)
   REAL XCLOUD(100), YCLOUD(100), ZCLOUD(100)
   REAL XOVLAP(500), YOVLAP(500), ZOVLAP(500)
C
C  FIND THE VOLUME OF THE PARALLELEPIPED
C
   INTER = 0
   INUM = 8
   CALL BOX(XARM,INUM,XS1,XL1)
   CALL BOX(YARM,INUM,YS1,YL1)
   CALL BOX(ZARM,INUM,ZS1,ZL1)
   V0=(XL1-XS1)*(YL1-YS1)*(ZL1-ZS1)
C
C  CHECK BOXES
C
   CALL TB(MF, NF, MA, NA, CONTER)
   IF(CONTER.EQ.0) GO TO 500
C
```

```
C  FIND THE INTERFERED POINTS
C
   CALL CH12(XPATH,YPATH,ZPATH,X2,Y2,Z2)
   CALL TF(MF,NF,ISTEP,V0,X2,Y2,Z2,CONTER,INTER,CHCKAL)
500        RETURN
   END
C
C  SUBROUTINE TO PLOT THE CLOUD OF POINTS
C
   SUBROUTINE PLTCLD (XCLOUD,YCLOUD,ZCLOUD,INTER,PLTINT)
C
C  X, Y and ZCLOUD are arrays that containt the x, y and z values of
C  the points to be plotted.  INTER is the number of points to be plotted.
C  PLTINT is a logical to tell if the points are to be plotted or not.
C  If PLTINT is false then is routine just rings the bell and returns.
C
   LOGICAL*1 PLTINT
   REAL XCLOUD(100), YCLOUD(100), ZCLOUD(100)
C
C  RING BELL
C
   CALL RINGBE
C
C  IF THE POINTS ARE NOT TO BE PLOTTED RETURN
C
   IF (.NOT. PLTINT) RETURN

   call pushat
   call color(i2(1))
C
C  DRAW X'S AT EACH POINT IN THE CLOUD
C
   delt=1.
   DO 100 I = 1, INTER
      call move(xcloud(i)-delt,ycloud(i),zcloud(i))
      call draw(xcloud(i)+delt,ycloud(i),zcloud(i))
      call move(xcloud(i),ycloud(i)-delt,zcloud(i))
      call draw(xcloud(i),ycloud(i)+delt,zcloud(i))
      call move(xcloud(i),ycloud(i),zcloud(i)-delt)
      call draw(xcloud(i),ycloud(i),zcloud(i)+delt)
100        CONTINUE
   call gflush

   call popatt
   RETURN
   END

c
c  FUNCTION
C
   INTEGER FUNCTION I2(I)
   INTEGER*2 M
   EQUIVALENCE (M,N)
   M=I
   I2=N
   RETURN
```

```
      END
```

```
C       program to detect if two parallelopiped intersect one another
C
C         Written by J. P. Duncan
C
C         Convert into FORTRAN 77 by Ken Law
C
C               May,  1985.
C
C       Variables used :
C
C       ARMX, ARMY, ARMZ        : co-ordinate of the robot arm
C        WPX,  WPY,  WPZ        : co-ordinate of the workpiece
C       MARM, NARM              : number of longitudes & latitudes in the arm
C        MWP,  NWP              : number of longitudes & latitudes in the WP
C       ARMX_OLD, ARMY_OLD,
C                      ARMZ_OLD : co-ordinates of the previous arm position
C       GRAPHIC_ON              : flag to state if graphic is required
C       STOP_AT1                : flag to go to next position if one
C                                        interference is found
C       INTER                   : flag to indicate one interference has occurred
C       ARM2WP                  : flag to show plane of arm to edge of workpiece
C       IT                      : parameter to calculate the position of arm
C
        IMPLICIT REALA8 (A-Z)
        CHARACTERA1 ANS
        CHARACTERA40 ARMDAT, WPDAT
        REALA8 ARMX(4,5), ARMY(4,5), ARMZ(4,5)
        REALA8 WPX(4,5),   WPY(4.5),  WPZ(4.5)
        REALA8 ARMX_OLD(4,5), ARMY_OLD(4.5), ARMZ OLD(4,5)
        REAL TIMLIM, DELTA, TIM
        LOGICALA2 INSIDE, GRAPHIC_ON, STOP_AT1, INTER, ARM2WP
        INTEGER NARM1, MARM1, NWP1, MWP1, IT, II, ILIM
        INTEGER  NARM,  MARM,  NWP,  MWP, LAST
C
        DATA MARM/4/,NARM/4/,MWP/4/,NWP/4/,LAST/20/
        DATA ROTX/0.0/,ROTY/0.0/,ROTZ/0.0/,DX/0.0/,DY/0.0/,DZ/0.0/
C
        COMMON /SP_FEAT/ GRAPHIC_ON, STOP_AT1, INTER, ARM2WP, IT
C
        WRITE(5,21)
 21       FORMAT(' Enter pause length: (in seconds) ',$)
        READ(5,A) TIMLIM
C
        GRAPHIC_ON = .FALSE.
        WRITE(5,22)
 22       FORMAT(' Do you graphics? (Y/N) ',$)
        READ(5,23) ANS
 23       FORMAT(A1)
        IF ((ANS.EQ.'Y').OR.(ANS.EQ.'y')) GRAPHIC_ON = .TRUE.
C
        STOP_AT1 = .TRUE.
        WRITE(5,24)
 24       FORMAT(' Do you full report? (Y/N) ',$)
        READ(5,23) ANS
        IF ((ANS.EQ.'Y').OR.(ANS.EQ.'y')) STOP_AT1 = .FALSE.
```

```
C
        CALL ASSIGN(1,'BOX422.DAT')
        CALL ASSIGN(2,'BOX222.ROT')
        CALL ASSIGN(4,'ROBOT.RES')
C
C       read in data for the arm
C
        DO 30 I = 1,MARM
          DO 30 J = 1, NARM
            READ(1,A) ARMX(I,J), ARMY(I,J), ARMZ(I,J)
 30     CONTINUE
C
C       read in data for the workpiece
C
        DO 40 I = 1, MWP
          DO 40 J = 1, NWP
            READ(2,A) WPX(I,J), WPY(I,J), WPZ(I,J)
 40     CONTINUE
C
C       make last point the same as first point
C
        NARM1 = NARM + 1
        DO 50 I = 1, MARM
          ARMX(I,NARM1) = ARMX(I,1)
          ARMY(I,NARM1) = ARMY(I,1)
          ARMZ(I,NARM1) = ARMZ(I,1)
 50     CONTINUE
C
        NWP1 = NWP + 1
        DO 60 I = 1, MWP
          WPX(I,NWP1) = WPX(I,1)
          WPY(I,NWP1) = WPY(I,1)
          WPZ(I,NWP1) = WPZ(I,1)
 60     CONTINUE
C
C       find the box for WP
C
        CALL RANGE2D(MWP,NWP1,WPX,WPXMIN,WPXMAX)
        CALL RANGE2D(MWP,NWP1,WPY,WPYMIN,WPYMAX)
        CALL RANGE2D(MWP,NWP1,WPZ,WPZMIN,WPZMAX)
C
C       initialize graphics
C
        IF (.NOT. GRAPHIC_ON) GOTO 90
        CALL GRINIT
        CALL AXES
        CALL PLTBOX(WPX,WPY,WPZ,MWP,NWP1)
        CALL PLTBOX(ARMX,ARMY,ARMZ,MARM,NARM1)
C
C       loop 800 defines the path of the ARM
C
 90     DO 800 IT = 1, LAST
C
C       find the box for ARM
C
 100    CALL RANGE2D(MARM,NARM1,ARMX,ARXMIN,ARXMAX)
        CALL RANGE2D(MARM,NARM1,ARMY,ARYMIN,ARYMAX)
```

```
          CALL RANGE2D(MARM,NARM1,ARMZ,ARZMIN,ARZMAX)
C
C         check if boxes overlap
C
          IF (.NOT. INSIDE(WPXMIN,WPXMAX,ARXMIN,ARXMAX)) GOTO 300
          IF (.NOT. INSIDE(WPYMIN,WPYMAX,ARYMIN,ARYMAX)) GOTO 300
          IF (.NOT. INSIDE(WPZMIN,WPZMAX,ARZMIN,ARZMAX)) GOTO 300
C
C         boxes overlap: check if edges pentrate planes
C         check plane of workpiece to edges of arm
C
  200     INTER = .FALSE.
          ARM2WP = .FALSE.
          CALL INTERF(MARM,NARM,ARMX,ARMY,ARMZ,MWP,NWP,WPX,WPY,WPZ)
          IF (INTER .AND. STOP_AT1) GOTO 300
C
C         check plane of arm to edges of workpiece
C
          ARM2WP = .TRUE.
          CALL INTERF(MWP,NWP,WPX,WPY,WPZ,MARM,NARM,ARMX,ARMY,ARMZ)
C
C         no interference: remove old and plot new position
C
  300     IF (IT .EQ. LAST) GOTO 800
          ROTX = ROTX + FLOAT(IT)
          ROTY = ROTY + FLOAT(IT)
          ROTZ = ROTZ + FLOAT(IT)
          DX   = DX   + FLOAT(IT)/10.
          DY   = DY   + FLOAT(ITAIT)/40.
          DZ   = DZ   + FLOAT(ITAITAIT)/800.
C
C         save present position for erasing
C
          DO 400 I = 1, MARM
            DO 400 J = 1, NARM1
              ARMX_OLD(I,J) = ARMX(I,J)
              ARMY_OLD(I,J) = ARMY(I,J)
              ARMZ_OLD(I,J) = ARMZ(I,J)
  400     CONTINUE
C
C         find new position of the arm
C
          CALL ROT(MARM,NARM1,ARMX,ARMY,ARMZ,XROT,YROT,ZROT)
          CALL TRN(ARMX,MARM,NARM1,DX)
          CALL TRN(ARMY,MARM,NARM1,DY)
          CALL TRN(ARMZ,MARM,NARM1,DZ)
C
C         erase old position and plot new position
C
          IF (.NOT. GRAPHIC_ON) GOTO 800
          CALL REMOVE
          CALL PLTBOX(ARMX_OLD,ARMY_OLD,ARMZ_OLD,MARM,NARM1)
          CALL APPEAR
          CALL PLTBOX(ARMX,ARMY,ARMZ,MARM,NARM1)
C
C         pause before moving to next position
```

```
ROBOT$MAIN                                                          28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56              Page   4
                                                                    28-May-1985 11:11:27    DSK0:[LAW.ROBOT]ROBOT.FOR;3

0172                TIM = SECNDS(0.0)
0173        795     DELTA = SECNDS(TIM)
0174                IF (DELTA .LT. TIMLIM) GOTO 795
0175        800     CONTINUE
0176      C
0177                WRITE(5,*)' Program terminated'
0178                STOP
0179                END


PROGRAM SECTIONS

    Name                                   Bytes   Attributes

  0 $CODE                                   1212   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                   167   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                  2348   PIC CON REL LCL NOSHR NOEXE  RD   WRT QUAD
  3 SP_FEAT                                   12   PIC OVR REL GBL   SHR NOEXE   RD   WRT LONG

    Total Space Allocated                   3739

ENTRY POINTS

    Address   Type  Name

  0-00000000        ROBOT$MAIN

VARIABLES

    Address   Type  Name                    Address   Type  Name                   Address   Type  Name                   Address   Type  Name

  2-000005A0  CHAR ANS                   3-00000006  L*2  ARM2WP                 2-000005A1  CHAR ARMDAT               2-00000670  R*8  ARXMAX
  2-00000668  R*8  ARXMIN                2-00000680  R*8  ARYMAX                 2-00000678  R*8  ARYMIN               2-00000690  R*8  ARZMAX
  2-00000688  R*8  ARZMIN                2-000006B4  R*4  DELTA                  2-00000610  R*8  DX                   2-00000618  R*8  DY
  2-00000620  R*8  DZ                    3-00000000  L*2  GRAPHIC_ON             2-00000628  R*8  I                    2-000006CC  I*4  II
  2-000006D0  I*4  ILIM                  3-00000004  L*2  INTER                  3-00000008  I*4  IT                   2-00000630  R*8  J
  2-000006E4  I*4  LAST                  2-000006D8  I*4  MARM                   2-000006C0  I*4  MARM1                2-000006E0  I*4  MWP
  2-000006C8  I*4  MWP1                  2-000006D4  I*4  NARM                   2-000006BC  I*4  NARM1                2-000006DC  I*4  NWP
  2-000006C4  I*4  NWP1                  2-000005F8  R*8  ROTX                   2-00000600  R*8  ROTY                 2-00000608  R*8  ROTZ
  3-00000002  L*2  STOP_AT1              2-000006B8  R*4  TIM                    2-000006B0  R*4  TIMLIM               2-000005C9  CHAR WPDAT
  2-00000640  R*8  WPXMAX                2-00000638  R*8  WPXMIN                 2-00000650  R*8  WPYMAX               2-00000648  R*8  WPYMIN
  2-00000660  R*8  WPZMAX                2-00000658  R*8  WPZMIN                 2-00000698  R*8  XROT                 2-000006A0  R*8  YROT
  2-000006A8  R*8  ZROT

ARRAYS

    Address   Type  Name                 Bytes  Dimensions

  2-00000000  R*8  ARMX                    160  (4, 5)
  2-000003C0  R*8  ARMX_OLD                160  (4, 5)
  2-000000A0  R*8  ARMY                    160  (4, 5)
  2-00000460  R*8  ARMY_OLD                160  (4, 5)
```

```
   2-00000140  RAB  ARMZ                 160  (4, 5)
   2-00000500  RAB  ARMZ_OLD             160  (4, 5)
   2-000001E0  RAB  WPX                  160  (4, 5)
   2-00000280  RAB  WPY                  160  (4, 5)
   2-00000320  RAB  WPZ                  160  (4, 5)

LABELS

    Address   Label      Address   Label      Address   Label      Address   Label      Address   Label      Address   Label

  1-00000043  21'      1-00000069  22'      1-00000085  23'      1-00000088  24'           **     30           **     40
       **     50            **     60       0-000002CE  90            **     100           **     200     0-00000356  300
       **     400      0-00000479  795      0-0000048E  800

FUNCTIONS AND SUBROUTINES REFERENCED

   Type  Name            Type  Name            Type  Name            Type  Name            Type  Name            Type  Name

         APPEAR                ASSIGN                AXES             RA4  FOR$SECNDS            GRINIT           LA2  INSIDE
         INTERF                PLTBOX                RANGE2D               REMOVE                ROT                   TRN
```

```
C
C
C       subroutine to check if any one edge of one parallelopiped pentrates
C       any one plane of another parallelopiped
C
        SUBROUTINE INTERF(MARM,NARM,ARMX,ARMY,ARMZ,MWP.NWP,WPX,WPY,WPZ)
C
        IMPLICIT REAL*8  (A-Z)
        INTEGER MARM, NARM, MWP, NWP
        REAL*8  ARMX(4,5), ARMY(4,5), ARMZ(4,5), WPX(4.5), WPY(4,5), WPZ(4,5)
        REAL*8 DIST, DET
        REAL*8 A1,B1,C1,P1,XA,YA,ZA,XB,YB,ZB,XC,YC,ZC
        INTEGER I, J, K, L, IT
        INTEGER NARM1, MARM1, NWP1, MWP1
        LOGICAL*2 FACE1, LINE1, GRAPHIC_ON, STOP_AT1, INTER, ARM2WP
C
        COMMON /SP_FEAT/ GRAPHIC_ON, STOP_AT1, INTER, ARM2WP, IT
        COMMON /PLANE/ A1,B1,C1,P1,XA,YA,ZA,XB,YB,ZB,XC,YC,ZC,I,J,FACE1
C
C       Variables used:
C         A1, B1, C1, P1 : defines the direction  cosines of a plane
C         XA, YA, ZA     : defines first  of three points of a plane
C         XB, YB, ZB     : defines second of three points of a plane
C         XC, YC, ZC     : defines third  of three points of a plane
C          I,  J         : defines the hor. and vert. indices for the
C                             'map of the world' on the workpiece
C          K,  L         : defines the hor. and vert. indices for the
C                             'map of the world' on the arm
C         FACE1          : flag indicating which triangle the program
C                             is working on. TRUE means triangle ABC.
C                              FALSE means triangle ACD.
C         LINE1          : flag indicating which edge the program is
C                             working on. TRUE means edge AB.
C                              FALSE means edge AD.
C         LBA, MBA, NBA  : direction cosine of the edge AB
C         LDA, MDA, NDA  : direction cosine of the edge AD
C
C       plane of WP to edge of ARM
C
        MWP1  = MWP  - 1
        NWP1  = NWP  - 1
        NARM1 = NARM - 1
        MARM1 = MARM - 1
C
        DO 1000 I = 1, MWP1
          DO 1000 J = 1, NWP
            XA = WPX(I,J)
            YA = WPY(I,J)
            ZA = WPZ(I,J)
            XB = WPX(I,J+1)
            YB = WPY(I,J+1)
            ZB = WPZ(I,J+1)
            XC = WPX(I+1,J+1)
            YC = WPY(I+1,J+1)
            ZC = WPZ(I+1,J+1)
            XD = WPX(I+1,J)
            YD = WPY(I+1,J)
```

```
              ZD = WPZ(I+1,J)
      C
      C     if I=1 do not find direction cosine for face 1
      C
            IF (I .EQ. 1) GOTO 950
              FACE1 = .TRUE.
      C
      C     find co-efficient of the plane
      C
      100   CONTINUE
              AN1 =  DET(1.,YA,ZA,1.,YB,ZB,1.,YC,ZC)
              BN1 = -DET(XA,1.,ZA,XB,1.,ZB,XC,1.,ZC)
              CN1 =  DET(XA,YA,1.,XB,YB,1.,XC,YC,1.)
              PN1 = -DET(XA,YA,ZA,XB,YB,ZB,XC,YC,ZC)
      C
              D   =  DSQRT(AN1*AN1+BN1*BN1+CN1*CN1)
      C
      C     find direction cosine of the plane
      C
              A1 = AN1 / D
              B1 = BN1 / D
              C1 = CN1 / D
              P1 = PN1 / D
      C
      C     for each edge, check if edge pentrate plane
      C
            DO 900 K = 1,NARM1
              DO 900 L = 1, NARM
                XAA = ARMX(K,L)
                YAA = ARMY(K,L)
                ZAA = ARMZ(K,L)
                XBA = ARMX(K,L+1)
                YBA = ARMY(K,L+1)
                ZBA = ARMZ(K,L+1)
                XDA = ARMX(K+1,L)
                YDA = ARMY(K+1,L)
                ZDA = ARMZ(K+1,L)
      C
      C     for (K,L) to (K,L+1)
      C
      C     if first row: all points are the same, so ignore
      C
              IF (K.EQ.1) GOTO 500
      C
              DBA = DIST(XAA,YAA,ZAA,XBA,YBA,ZBA)
              LBA = (XBA - XAA)/ DBA
              MBA = (YBA - YAA)/ DBA
              NBA = (ZBA - ZAA)/ DBA
      C
      C     for edge AB
      C
            LINE1 = .TRUE.
            INTER = .FALSE.
            CALL PENTR(LBA,MBA,NBA,XAA,YAA,ZAA,XBA,YBA,ZBA,K,L,LINE1,INTER)
            IF (STOP_AT1 .AND. INTER) GOTO 9000
      C
      C     fo       L)       K+1
```

```
C
 500    CONTINUE
        LINE1 = .FALSE.
        IF (K.GE.MARM1) GOTO 900
             DDA = DIST(XAA,YAA,ZAA,XDA,YDA,ZDA)
             LDA = (XDA - XAA)/ DDA
             MDA = (YDA - YAA)/ DDA
             NDA = (ZDA - ZAA)/ DDA
C
C       for edge DA
C
        INTER = .FALSE.
        CALL PENTR(LDA,MDA,NDA,XAA,YAA,ZAA,XDA,YDA,ZDA,K,L,LINE1,INTER)
        IF(STOP_AT1 .AND. INTER) GOTO 9000
C
 900    CONTINUE
C
        IF (FACE1 .NE. .TRUE.) GOTO 1000
C
C       do face 2
C
C       if I=MWP1, do not find direction cosine for face 2
C
        IF (I .EQ. MWP1) GOTO 1000
 950    FACE1 = .FALSE.
        XB = XD
        YB = YD
        ZB = ZD
        GOTO 100
C
 1000   CONTINUE
C
 9000   RETURN
        END
```

```
ROBOT.LIS:1                      28-MAY-1985 11:18              Page 9

INTERF                                                          28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56                Page   9
                                                                28-May-1985 11:11:27    DSKO:[LAW.ROBOT]ROBOT.FOR:3

PROGRAM SECTIONS

    Name                                    Bytes   Attributes

  0 $CODE                                     808   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                      4   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                    688   PIC CON REL LCL NOSHR NOEXE   RD   WRT QUAD
  3 SP_FEAT                                    12   PIC OVR REL GBL   SHR NOEXE   RD   WRT LONG
  4 PLANE                                     114   PIC OVR REL GBL   SHR NOEXE   RD   WRT LONG

    Total Space Allocated                    1626

ENTRY POINTS

    Address   Type  Name

  0-00000000        INTERF

VARIABLES

    Address   Type  Name              Address   Type  Name              Address   Type  Name              Address   Type  Name

  4-00000000  R*8   A1            2-00000020  R*8   AN1           3-00000006  L*2   ARM2WP        4-00000008  R*8   B1
  2-00000028  R*8   BN1           4-00000010  R*8   C1            2-00000030  R*8   CN1           2-00000040  R*8   D
  2-00000090  R*8   DBA           2-000000B0  R*8   DDA           4-00000070  L*2   FACE1         3-00000000  L*2   GRAPHIC_ON
  4-00000068  I*4   I             3-00000004  L*2   INTER         3-00000008  I*4   IT            4-0000006C  I*4   J
  2-000000D0  I*4   K             2-000000D4  I*4   L             2-00000098  R*8   LBA           2-000000B8  R*8   LDA
  3-00000000  L*2   LINE1        AP-00000004@ I*4   MARM          2-000000DC  I*4   MARM1         2-000000A0  R*8   MBA
  2-000000C0  R*8   MDA          AP-00000018@ I*4   MWP           2-000000E4  I*4   MWP1         AP-00000008@ I*4   NARM
  2-000000D8  I*4   NARM1         2-000000A8  R*8   NBA           2-000000C8  R*8   NDA          AP-0000001C@ I*4   NWP
  2-000000E0  I*4   NWP1          4-00000018  R*8   P1            2-00000038  R*8   PN1           3-00000002  L*2   STOP_AT1
  4-00000020  R*8   XA            2-00000048  R*8   XAA           4-00000038  R*8   XB            2-00000060  R*8   XBA
  4-00000050  R*8   XC            2-00000008  R*8   XD            2-00000078  R*8   XDA           4-00000028  R*8   YA
  2-00000050  R*8   YAA           4-00000040  R*8   YB            2-00000068  R*8   YBA           4-00000058  R*8   YC
  2-00000010  R*8   YD            2-00000080  R*8   YDA           4-00000030  R*8   ZA            2-00000058  R*8   ZAA
  4-00000048  R*8   ZB            2-00000070  R*8   ZBA           4-00000060  R*8   ZC            2-00000018  R*8   ZD
  2-00000088  R*8   ZDA

ARRAYS

    Address   Type  Name                    Bytes  Dimensions

 AP-0000000C@ R*8   ARMX                      160  (4, 5)
 AP-00000010@ R*8   ARMY                      160  (4, 5)
 AP-00000014@ R*8   ARMZ                      160  (4, 5)
 AP-00000020@ R*8   WPX                       160  (4, 5)
 AP-00000024@ R*8   WPY                       160  (4, 5)
 AP-00000028@ R*8   WPZ                       160  (4, 5)
```

```
INTERF                                                      28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56                  Page  10
                                                            28-May-1985 11:11:27    DSK0:[LAW.ROBOT]ROBOT.FOR;3

LABELS

    Address   Label        Address   Label        Address   Label        Address   Label        Address   Label        Address   Label

  0-0000011B  100        0-0000026D  500        0-000002C9  900        0-000002EF  950        0-0000030F  1000       0-00000327  9000

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name             Type  Name             Type  Name             Type  Name

   R*8  DET               R*8  DIST              R*8  MTH$DSQRT              PENTR
```

```
C
C
C        subroutine PENTR checks if edge pentrates the plane
C           and finds the point of pentration. If the point
C           is inside the plane defined by the 3 points,
C           interferece occurs. A message is printed.
C
         SUBROUTINE PENTR(LL,ML,NL,XO,YO,ZO,XL,YL,ZL,K,L,LINE1)
C
         IMPLICIT REAL*8 (A-Z)
         LOGICAL*2 LINE1, FACE1, GRAPHIC_ON, STOP_AT1, INTER, ARM2WP
         INTEGER K, L, I, J, IT
         REAL*8 A1,B1,C1,P1,XA,YA,ZA,XB,YB,ZB,XC,YC,ZC
C
         COMMON /SP_FEAT/ GRAPHIC_ON, STOP_AT1, INTER, ARM2WP, IT
         COMMON /PLANE/ A1,B1,C1,P1,XA,YA,ZA,XB,YB,ZB,XC,YC.ZC,I,J,FACE1
C
         NUM = -(A1*XO+B1*YO+C1*ZO+P1)
         DEN =  A1*LL + B1*ML + C1*NL
         IF (DABS(NUM) .LT. 0.00001) GOTO 100
 50      IF (DABS(DEN) .LT. 0.00001) GOTO 200
         U = NUM / DEN
         XI = XO + U*LL
         YI = YO + U*ML
         ZI = ZO + U*NL
C
C        check if (XI,YI,ZI) is between (XO,YO,ZO) and (XL,YL,ZL)
C
C------ IF (XL .GT. XO) GOTO 90
C
C        XL .LT. XO
C
C------ IF ((XI .GE. XL) .AND. (XI .LE. XO)) GOTO 1000
C------   GOTO 80
C
C---- 90        IF ((XI .GE. XO) .AND. (XI .LE. XL)) GOTO 1000
C
         DOL = DIST(XO,YO,ZO,XL,YL,ZL)
         DIL = DIST(XI,YI,ZI,XL,YL,ZL)
         DIO = DIST(XI,YI,ZI,XO,YO,ZO)
         IF (DABS(DIL+DIO-DOL) .LT. 0.01) GOTO 1000
 80      RETURN
C
C        NUM = 0.0
C
 100     CONTINUE
C------ CALL MSG(I,J,FACE1,K,L,LINE1,IT,ARM2WP)
C------ WRITE(4,*)' First point is coincident'
         GOTO 50
C
C        DEN = 0.0
C
 200     CONTINUE
C------ CALL MSG(I,J,FACE1,K,L,LINE1,IT,ARM2WP)
C------ WRITE(4,*)' EDGE IS PARALLEL'
         RETURN
```

```
C       find the point of intersection within triangular facet
C
 1000   AR     = AREA(XA,YA,ZA,XB,YB,ZB,XC,YC,ZC)
        ARABC =  DABS(AR)
        AR     = AREA(XI,YI,ZI,XB,YB,ZB,XC,YC,ZC)
        ARIBC =  DABS(AR)
        AR     = AREA(XI,YI,ZI,XA,YA,ZA,XB,YB,ZB)
        ARIAB =  DABS(AR)
        AR     = AREA(XI,YI,ZI,XA,YA,ZA,XC,YC,ZC)
        ARIAC =  DABS(AR)
C
        ARTOT = ARIAB + ARIBC + ARIAC
        IF (DABS(ARABC-ARTOT) .GT. 0.00001) RETURN
C
C       (XI,YI,ZI) is within triangle
C
        INTER = .TRUE.
        IF (GRAPHIC_ON) CALL PLOT_CLOUD(XI,YI,ZI)
        CALL MSG(I,J,FACE1,K,L,LINE1,IT,ARM2WP)
        WRITE(4,1100)XI,YI,ZI
 1100     FORMAT(' Intersection within facet at ',3F10.4)
        RETURN
        END
```

```
PROGRAM SECTIONS

   Name                                   Bytes   Attributes

 0 $CODE                                    535   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
 1 $PDATA                                    38   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
 2 $LOCAL                                   388   PIC CON REL LCL NOSHR NOEXE   RD   WRT QUAD
 3 SP_FEAT                                   12   PIC OVR REL GBL   SHR NOEXE   RD   WRT LONG
 4 PLANE                                    114   PIC OVR REL GBL   SHR NOEXE   RD   WRT LONG

   Total Space Allocated                   1087


ENTRY POINTS

   Address   Type   Name

 0-00000000         PENTR


VARIABLES

   Address   Type  Name                Address   Type  Name                Address   Type  Name                Address   Type  Name

 4-00000000  R*8   A1               2-00000048  R*8   AR                3-00000050  R*8   ARABC             2-00000060  R*8   ARIAB
 2-00000068  R*8   ARIAC            2-00000058  R*8   ARIBC             3-00000006  L*2   ARM2WP            2-00000070  R*8   ARTOT
 4-00000008  R*8   B1               4-00000010  R*8   C1                2-00000008  R*8   DEN               2-00000038  R*8   DIL
 2-00000040  R*8   DIO              2-00000030  R*8   DOL               4-00000070  L*2   FACE1             3-00000000  L*2   GRAPHIC_ON
 4-00000068  I*4   I                3-00000004  L*2   INTER             3-00000008  I*4   IT                4-0000006C  I*4   J
AP-00000028@ I*4   K               AP-0000002C@ I*4   L                AP-00000030@ L*2   LINE1            AP-00000004@ R*8   LL
```

```
PENTR

 AP-00000008@ R*8  ML           AP-0000000C@ R*8  NL           2-00000000  R*8  NUM          4-00000018  R*8  P1
  3-00000002  L*2  STOP_AT1      2-00000010  R*8  U            4-00000020  R*8  XA           4-00000038  R*8  XB
  4-00000050  R*8  XC            2-00000018  R*8  XI          AP-0000001C@ R*8  XL          AP-00000010@ R*8  XD
  4-00000028  R*8  YA            4-00000040  R*8  YB           4-00000058  R*8  YC           2-00000020  R*8  YI
 AP-00000020@ R*8  YL          AP-00000014@ R*8  YO            4-00000030  R*8  ZA           4-00000048  R*8  ZB
  4-00000060  R*8  ZC            2-00000028  R*8  ZI          AP-00000024@ R*8  ZL          AP-00000018@ R*8  ZO

LABELS

   Address   Label        Address   Label        Address   Label        Address   Label        Address   Label        Address   Label

 0-0000005F  50             **      80        0-0000012A  100        0-0000012D  200        0-0000012E  1000       1-00000000  1100'

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name            Type  Name            Type  Name            Type  Name

   R*8  AREA             R*8  DIST                  MSG                   PLOT_CLOUD
```

```
C
C
C       function det finds the determinant of a 3 by 3 matrix
C                            |X1  Y1  Z1|
C         whose points are |X2  Y2  Z2|
C                            |X3  Y3  Z3|
C
        REAL*8 FUNCTION DET(X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3)
C
        IMPLICIT REAL*8 (A-Z)
C
        DET1 = (X1*Y2*Z3) + (Y1*Z2*X3) + (Z1*X2*Y3)
        DET2 = (Z1*Y2*X3) + (X1*Z2*Y3) + (Y1*X2*Z3)
        DET  = DET1 - DET2
        RETURN
        END


PROGRAM SECTIONS

    Name                                       Bytes   Attributes

  0 $CODE                                         90   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  2 $LOCAL                                        24   PIC CON REL LCL NOSHR NOEXE   RD   WRT QUAD

    Total Space Allocated                        114

ENTRY POINTS

    Address  Type  Name

  0-00000000  R*8  DET

VARIABLES

    Address  Type  Name                 Address  Type  Name                 Address  Type  Name                 Address  Type  Name

  2-00000008  R*8  DET1             2-00000010  R*8  DET2            AP-00000004@ R*8  X1             AP-00000010@ R*8  X2
 AP-0000001C@ R*8  X3            AP-00000008@ R*8  Y1            AP-00000014@ R*8  Y2             AP-00000020@ R*8  Y3
 AP-0000000C@ R*8  Z1            AP-00000018@ R*8  Z2            AP-00000024@ R*8  Z3
```

```
ROBOT.LIS;1                  28-MAY-1985 11:10              Page 15

                                                            28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56              Page 15
                                                            28-May-1985 11:11:37    DSK0:[LAW.ROBOT]ROBOT.FOR;3

0001      C
0002      C
0003      C         function dist finds the distance between two points
0004      C           whose co-ordinates are (X1,Y1,Z1), (X2,Y2,Z2)
0005      C
0006                REALA8 FUNCTION DIST(X1,Y1,Z1,X2,Y2,Z2)
0007      C
0008                IMPLICIT REALA8 (A-Z)
0009      C
0010                D = (X2-X1)A(X2-X1) + (Y2-Y1)A(Y2-Y1) + (Z2-Z1)A(Z2-Z1)
0011                DIST = DSQRT(D)
0012                RETURN
0013                END


PROGRAM SECTIONS

   Name                                   Bytes   Attributes

 0 $CODE                                    59    PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
 2 $LOCAL                                   16    PIC CON REL LCL NOSHR NOEXE   RD   WRT QUAD

   Total Space Allocated                    75

ENTRY POINTS

   Address  Type  Name

 0-00000000  RA8  DIST

VARIABLES

   Address  Type  Name               Address  Type  Name               Address  Type  Name               Address  Type  Name

 2-00000008  RA8  D             AP-00000004@ RA8  X1             AP-00000010@ RA8  X2             AP-00000008@ RA8  Y1
AP-00000014@ RA8  Y2            AP-0000000C@ RA8  Z1             AP-00000018@ RA8  Z2

FUNCTIONS AND SUBROUTINES REFERENCED

 Type  Name

  RA8  MTH$DSQRT
```

```
C
C
C       subroutine MSG prints the message of which edge pentrates which plane
C
        SUBROUTINE MSG(I,J,FACE1,K,L,LINE1,IT,ARM2WP)
C
        INTEGER I,J,K,L,IT
        LOGICAL*2 FACE1,LINE1,ARM2WP
C
        IF (ARM2WP) WRITE(4.10)IT
 10       FORMAT(' At T=',I4,'  Plane of ARM to edge of  WP')
        IF (.NOT. ARM2WP) WRITE(4.20) IT
 20       FORMAT(' At T= ',I4,'  Plane of  WP to edge of ARM')
        IF (FACE1) GOTO 1000
        IF (LINE1) GOTO  500
C
C       face2 and vertical -> plane ACD and edge AD
C
        WRITE(4,100)I,J,K,L
 100      FORMAT(' I,J=', 2I4,' plane ACD and K,L=',2I4,' edge AD')
        GOTO 9999
C
C       face2 and horizontal -> plane ACD and edge AB
C
 500    WRITE(4,510)I,J,K,L
 510      FORMAT(' I,J=',2I4,' plane ACD and K,L=',2I4,' edge AB')
        GOTO 9999
C
 1000   IF (LINE1) GOTO 1500
C
C       face1 and vertical edge -> plane ABC and edge AD
C
        WRITE(4,1010)I,J,K,L
 1010     FORMAT(' I,J=',2I4,' plane ABC and K,L=',2I4,' edge AD')
        GOTO 9999
C
C       face1 and horizontal edge -> plane ABC and edge AB
C
 1500   WRITE(4,1510)I,J,K,L
 1510     FORMAT(' I,J=', 2I4,' plane ABC and K,L=',2I4,' edge AB')
C
 9999   RETURN
        END
```

```
MSG                                                            28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56              Page  17
                                                               28-May-1985 11:11:27    DSK0:[LAW.ROBOT]ROBOT.FOR;3

PROGRAM SECTIONS

    Name                                   Bytes   Attributes

  0 $CODE                                    351   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                   273   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG

    Total Space Allocated                    624

ENTRY POINTS

    Address   Type  Name

  0-00000000        MSG

VARIABLES

    Address   Type  Name                Address   Type  Name                Address   Type  Name                Address   Type  Name

 AP-00000020@ LA2   ARM2WP           AP-0000000C@ LA2   FACE1            AP-00000004@ IA4   I                AP-0000001C@ IA4   IT
 AP-00000008@ IA4   J                AP-00000010@ IA4   K                AP-00000014@ IA4   L                AP-00000018@ LA2   LINE1

LABELS

    Address   Label       Address   Label       Address   Label       Address   Label       Address   Label       Address   Label

  1-00000000  10'      1-0000002A  20'      1-00000055  100'     0-0000009A  500      1-00000084  510'     0-000000DB  1000
  1-000000B3  1010'    0-00000120  1500     1-000000E2  1510'    0-0000015E  9999
```

```
0001    C
0002    C
0003    C       function area finds the area of a triangle given the
0004    C         three vertices : (X1,Y1,Z1), (X2,Y2,Z2), (X3,Y3,Z3)
0005    C
0006            REAL*8 FUNCTION AREA(X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3)
0007    C
0008            IMPLICIT REAL*8 (A-Z)
0009    C
0010            L  = (X2-X1)*(X2-X1) + (Y2-Y1)*(Y2-Y1) + (Z2-Z1)*(Z2-Z1)
0011            LG12 = DSQRT(L)
0012            L  = (X3-X1)*(X3-X1) + (Y3-Y1)*(Y3-Y1) + (Z3-Z1)*(Z3-Z1)
0013            LG13 = DSQRT(L)
0014            L  = (X3-X2)*(X3-X2) + (Y3-Y2)*(Y3-Y2) + (Z3-Z2)*(Z3-Z2)
0015            LG23 = DSQRT(L)
0016            S123 = (LG12 + LG23 + LG13)/2.
0017    C
0018            AR = S123*(S123-LG12)*(S123-LG23)*(S123-LG13)
0019            AREA = DSQRT(AR)
0020            RETURN
0021            END


PROGRAM SECTIONS

    Name                                    Bytes   Attributes

  0 $CODE                                     224   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  2 $LOCAL                                     56   PIC CON REL LCL NOSHR NOEXE   RD   WRT QUAD

    Total Space Allocated                     280


ENTRY POINTS

    Address   Type  Name

  0-00000000  R*8   AREA


VARIABLES

    Address   Type  Name          Address   Type  Name          Address   Type  Name          Address   Type  Name

  2-00000030  R*8   AR          2-00000008  R*8   L           2-00000010  R*8   LG12        2-00000018  R*8   LG13
  2-00000020  R*8   LG23        2-00000028  R*8   S123       AP-00000004@ R*8   X1         AP-00000010@ R*8   X2
 AP-0000001C@ R*8   X3         AP-00000008@ R*8   Y1         AP-00000014@ R*8   Y2         AP-00000020@ R*8   Y3
 AP-0000000C@ R*8   Z1         AP-00000018@ R*8   Z2         AP-00000024@ R*8   Z3
```

```
FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name

   R*8  MTH$DSQRT
```

```
C
C        subroutine GRINIT initializes Interactive Graphics Language (IGL)
C          plotting routines. All the calling routines are defined in IGL.
C
         SUBROUTINE GRINIT
C
         REAL*8 XSCALE, XROT, YROT, ZROT
         REAL*8 MINX,MAXX,MINY,MAXY,MINZ,MAXZ
         DATA XROT/30.0/, YROT/10.0/, ZROT/90.0/,XSCALE/3.0/
         DATA MINX/-50.0/, MAXX/10.0/,
         1    MINY/-10.0/, MAXY/50.0/,
         2    MINZ/-30.0/, MAXZ/30.0/
C
         ITERM_TYPE = 4027
         CALL GRSTRT(0,1)
         CALL NEWDEV(ITERM_TYPE,1)
C
C        default text quality = 4, but explicitly setted anyways
C
         CALL TXQUAL(4)
C
C
         CALL WIND3D(MINX,MAXX,MINY,MAXY,MINZ,MAXZ)
         CALL VWPT3D(MINX,MAXX,MINY,MAXY,MINZ,MAXZ)
         CALL ROTXYZ(XROT,YROT,ZROT)
         CALL SCAXYZ(XSCALE,XSCALE,XSCALE)
         RETURN
         END
```

```
GRINIT                                                  28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56              Page  21
                                                        28-May-1985 11:11:27    DSK0:[LAW.ROBOT]ROBOT.FOR;3

PROGRAM SECTIONS

    Name                                Bytes   Attributes

  0 $CODE                                  74   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                 12   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                176   PIC CON REL LCL NOSHR NOEXE   RD   WRT QUAD

    Total Space Allocated                 262

ENTRY POINTS

    Address   Type  Name

  0-00000000        GRINIT

VARIABLES

    Address   Type  Name                 Address   Type  Name                 Address   Type  Name                 Address   Type  Name

  2-00000050  I*4   ITERM_TYPE         2-00000028  R*8   MAXX               2-00000038  R*8   MAXY               2-00000048  R*8   MAXZ
  2-00000020  R*8   MINX               2-00000030  R*8   MINY               2-00000040  R*8   MINZ               2-00000008  R*8   XROT
  2-00000000  R*8   XSCALE             2-00000010  R*8   YROT               2-00000018  R*8   ZROT

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name              Type  Name              Type  Name              Type  Name              Type  Name              Type  Name

        GRSTRT                  NEWDEV                  ROTXYZ                  SCAXYZ                  TXQUAL                  VWPT3D
        WIND3D
```

```
                                                                 28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56           Page  22
                                                                 28-May-1985 11:11:27    DSK0:[LAW.ROBOT]ROBOT.FOR;3
0001    C
0002    C       function INSIDE checks if X1 and X2 are within the MIN and MAX
0003    C
0004            LOGICAL*2 FUNCTION INSIDE(MIN,MAX,X1,X2)
0005    C
0006            REAL*8 MIN,MAX,X1,X2
0007    C
0008            INSIDE = .FALSE.
0009            IF ((X1.LT.MIN) .OR. (X1.GT.MAX)) GOTO 100
0010              GOTO 200
0011     100    IF ((X2.LT.MIN) .OR. (X2.GT.MAX)) GOTO 300
0012     200      INSIDE = .TRUE.
0013     300    RETURN
0014            END


PROGRAM SECTIONS

    Name                                    Bytes   Attributes

  0 $CODE                                      51   PIC CON REL LCL   SHR   EXE    RD NOWRT LONG
  2 $LOCAL                                      4   PIC CON REL LCL NOSHR NOEXE    RD   WRT LONG

    Total Space Allocated                      55

ENTRY POINTS

    Address  Type  Name

  0-00000000  L*2  INSIDE

VARIABLES

    Address  Type  Name                Address  Type  Name                Address  Type  Name                Address  Type  Name

 AP-00000008@ R*8  MAX             AP-00000004@ R*8  MIN             AP-0000000C@ R*8  X1              AP-00000010@ R*8  X2

LABELS

    Address   Label       Address   Label       Address   Label

  0-0000001C  100       0-0000002A  200       0-0000002E  300
```

```
C
C       subroutine PLTBOX plots a box without any hidden line removal.
C         It uses IGL's DRAW3D and MOVE3D plotting routines.
C
        SUBROUTINE PLTBOX(X,Y,Z,M,N)
C
        REAL*8 X(M,N), Y(M,N), Z(M,N)
C
        CALL MOVE3D(X(2,1),Y(2,1),Z(2,1))
C
        DO 10 J = 2,N
 10       CALL DRAW3D(X(2,J),Y(2,J),Z(2,J))
C
        CALL DRAW3D(X(3,1),Y(3,1),Z(3,1))
C
        DO 20 J = 2,N
 20       CALL DRAW3D(X(3,J),Y(3,J),Z(3,J))
C
        DO 30 J = 2,N-1
          CALL MOVE3D(X(2,J),Y(2,J),Z(2,J))
 30       CALL DRAW3D(X(3,J),Y(3,J),Z(3,J))
        RETURN
        END


PROGRAM SECTIONS

    Name                                 Bytes   Attributes

  0 $CODE                                  405   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  2 $LOCAL                                 176   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                  581

ENTRY POINTS

    Address   Type   Name

  0-00000000         PLTBOX

VARIABLES

    Address   Type   Name              Address   Type  Name              Address   Type  Name

  2-00000000  I*4    J              AP-00000010@ I*4   M              AP-00000014@ I*4   N
```

```
PLTBOX                                                    28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56          Page  24
                                                          28-May-1985 11:11:27    DSKO:[LAW.ROBOT]ROBOT.FOR:3

ARRAYS

    Address   Type  Name                  Bytes  Dimensions

 AP-00000004@ RA8   X                       **   (*, *)
 AP-00000008@ RA8   Y                       **   (*, *)
 AP-0000000C@ RA8   Z                       **   (*, *)

LABELS

    Address   Label        Address   Label        Address   Label

      **      10             **      20             **      30

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name              Type  Name

        DRAW3D                  MOVE3D
```

```
ROBOT.LIS;1                28-MAY-1985 11:18              Page 25

                                                          28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56            Page  25
                                                          28-May-1985 11:11:27    DSK0:[LAW.ROBOT]ROBOT.FOR:3
0001    C
0002    C       subroutine to find max and min of a 2-D array
0003    C
0004            SUBROUTINE RANGE2D(M,N,A,MIN,MAX)
0005    C
0006            REAL*8 A(M,N), MIN, MAX
0007    C
0008            MIN = A(1,1)
0009            MAX = A(1,1)
0010            DO 40 I = 1,M
0011              DO 40 J = 1,N
0012                IF (A(I,J) - MIN) 10,20,20
0013       10         MIN = A(I,J)
0014       20       IF (A(I,J) - MAX) 40,40,30
0015       30         MAX = A(I,J)
0016       40   CONTINUE
0017            RETURN
0018            END



PROGRAM SECTIONS

   Name                                        Bytes   Attributes

 0 $CODE                                         131   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
 2 $LOCAL                                         60   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

   Total Space Allocated                         191

ENTRY POINTS

   Address  Type  Name

 0-00000000       RANGE2D

VARIABLES

   Address  Type  Name                Address  Type  Name                Address  Type  Name                Address  Type  Name

 2-00000000  I*4  I                2-00000004  I*4  J               AP-00000004@ I*4  M               AP-00000014@ R*8  MAX
AP-00000010@ R*8  MIN             AP-00000008@ I*4  N

ARRAYS

   Address  Type  Name                 Bytes  Dimensions

AP-0000000C@ R*8  A                      **  (*, *)
```

```
LABELS

    Address   Label       Address   Label        Address   Label       Address   Label

      **      10        0-00000067  20             **      30        0-00000073  40
```

```
C
        SUBROUTINE AXES
C
C       THIS SUBROUTINE DRAWS A 3-D AXES
C
C       This is one of the example routines in the IGL manual.
C       Only the length of the axes have been scaled.
C       All calling routines are defined in IGL manual.
C
        REAL XAXISX(6), XAXISY(6), XAXISZ(6), YAXISX(6),
        1    YAXISY(6), YAXISZ(6), ZAXISX(6), ZAXISY(6),
        2    ZAXISZ(6)
        DATA XAXISX/0.0,4.50,4.50,4.70,4.50,4.50/,
        1    XAXISY/0.0, 0.0, 1.0, 0.0,-1.0, 0.0/,
        2    XAXISZ/0.0, 0.0, 0.0, 0.0, 0.0, 0.0/,
        3    YAXISX/0.0, 0.0, 1.0, 0.0,-1.0, 0.0/,
        4    YAXISY/0.0,4.50,4.50,4.70,4.50,4.50/,
        5    YAXISZ/0.0, 0.0, 0.0, 0.0, 0.0, 0.0/,
        6    ZAXISX/0.0, 0.0,-1.0, 0.0, 1.0, 0.0/,
        7    ZAXISY/0.0, 0.0, 0.0, 0.0, 0.0, 0.0/,
        8    ZAXISZ/0.0,4.50,4.50,4.70,4.50,4.50/
C
C       OUTPUT THE X, Y AND Z AXES
C
C       X-AXES
C
        CALL MOVE3D(XAXISX(1), XAXISY(1), XAXISZ(1))
        DO 1 I= 2,6
                CALL DRAW3D (XAXISX(I), XAXISY(I), XAXISZ(I))
1       CONTINUE
C
C       CREATE A LABEL X FOR THE X-AXIS
C
        CALL MOVE3D(4.80,1.5,0.0)
        CALL DRAW3D(5.00,-1.5,0.0)
        CALL MOVE3D(5.00,1.5,0.0)
        CALL DRAW3D(4.80,-1.5,0.0)
C
C       Y-AXIS
C
        CALL MOVE3D(YAXISX(1), YAXISY(1), YAXISZ(1))
        DO 2 I= 2,6
                CALL DRAW3D (YAXISX(I), YAXISY(I), YAXISZ(I))
2       CONTINUE
C
C       CREATE A LABEL Y FOR THE Y AXIS
C
        CALL MOVE3D(-1.5,5.00,0.0)
        CALL DRAW3D(0.0,4.80,0.0)
        CALL MOVE3D(1.5,5.00,0.0)
        CALL DRAW3D(-1.5,4.70,0.0)
C
C       Z AXIS
C
        CALL MOVE3D (ZAXISX(1), ZAXISY(1), ZAXISZ(1))
        DO 3 I= 2,6
                CALL DRAW3D (ZAXISX(I), ZAXISY(I), ZAXISZ(I))
```

```
AXES                                                    28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56              Page  28
                                                        28-May-1985 11:11:27    DSK0:[LAW.ROBOT]ROBOT.FOR;3

0058      3       CONTINUE
0059      C
0060      C       CREATE A LABEL Z FOR THE Z-AXIS
0061      C
0062              CALL MOVE3D(-1.5,1.5,4.75)
0063              CALL DRAW3D(1.5,1.5,4.75)
0064              CALL DRAW3D(-1.5,-1.5,4.75)
0065              CALL DRAW3D(1.5,-1.5,4.75)
0066              RETURN
0067              END


PROGRAM SECTIONS

    Name                                   Bytes   Attributes

  0 $CODE                                    244   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                    28   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                   476   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                    748

ENTRY POINTS

    Address  Type  Name

  0-00000000       AXES

VARIABLES

    Address  Type  Name

  2-000000D8  I*4  I

ARRAYS

    Address  Type  Name                  Bytes  Dimensions

  2-00000000  R*4  XAXISX                   24  (6)
  2-00000018  R*4  XAXISY                   24  (6)
  2-00000030  R*4  XAXISZ                   24  (6)
  2-00000048  R*4  YAXISX                   24  (6)
  2-00000060  R*4  YAXISY                   24  (6)
  2-00000078  R*4  YAXISZ                   24  (6)
  2-00000090  R*4  ZAXISX                   24  (6)
  2-000000A8  R*4  ZAXISY                   24  (6)
  2-000000C0  R*4  ZAXISZ                   24  (6)
```

```
LABELS

    Address   Label        Address   Label        Address   Label

      AA        1            AA        2            AA        3

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name             Type  Name

        DRAW3D                 MOVE3D
```

```
C
C       SUBROUTINE TO PLOT THE CLOUD OF POINTS
C
C        This routine was taken from Alan Steeves' version of the robot arm
C         with one modification, which is to plot a single point instead of
C          a series of points.
C
        SUBROUTINE PLOT_CLOUD (X_CLOUD,Y_CLOUD,Z_CLOUD)
C
C       X, Y and Z_cloud are variables that contains the x, y and z values of
C       the point to be plotted.
C
        REAL*8 X_CLOUD, Y_CLOUD, Z_CLOUD
        CHARACTER X_CHARACTER(1)
        DATA X_CHARACTER/'X'/
C
C       SET THE TEXT POSTION TO CENTER
C
        CALL TXICUR(5)
C
C       SET THE TEXT SIZE
C
        CALL TXSIZE(0, 2., 2.)
C
C       RING BELL
C
        CALL BELL
C
C       DRAW X'S AT EACH POINT IN THE CLOUD
C
                CALL MOVE3D(X_CLOUD, Y_CLOUD, Z_CLOUD)
                CALL TEXT(1, X_CHARACTER)
        RETURN
        END
```

```
PROGRAM SECTIONS

    Name                                     Bytes   Attributes

  0 $CODE                                       64   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                      16   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                      88   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                      168

ENTRY POINTS

    Address   Type  Name

  0-00000000        PLOT_CLOUD

VARIABLES

    Address   Type  Name                Address   Type  Name                Address   Type  Name

 AP-00000040 RA8  X_CLOUD          AP-00000080 RA8  Y_CLOUD          AP-000000C0 RA8  Z_CLOUD

ARRAYS

    Address   Type  Name                    Bytes  Dimensions

  2-00000000  CHAR X_CHARACTER                  1  (1)

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name             Type  Name             Type  Name             Type  Name             Type  Name

        BELL                   MOVE3D                 TEXT                   TXICUR                 TXSIZE
```

```
                                                        28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56              Page  32
                                                        28-May-1985 11:11:27    DSKO:[LAW.ROBOT]ROBOT.FOR:3

0001    C
0002    C       subroutine TRN translates the data by the amount DX
0003    C         given a 2-D array (named A) and dimensions M,N.
0004    C
0005            SURROUTINE TRN(A,M,N,DX)
0006    C
0007            REAL*8 A(M,N), DX
0008    C
0009            DO 10 I = 1,M
0010              DO 10 J = 1,N
0011     10         A(I,J) = A(I,J) + DX
0012            RETURN
0013            END


PROGRAM SECTIONS

    Name                                          Bytes   Attributes

  0 $CODE                                           91    PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  2 $LOCAL                                          60    PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                          151

ENTRY POINTS

    Address   Type  Name

  0-00000000        TRN

VARIABLES

    Address   Type  Name                 Address   Type  Name                 Address   Type  Name                 Address   Type  Name

 AP-00000010@ R*8   DX                 2-00000000  I*4   I                  2-00000004  I*4   J                  AP-00000008@ I*4   M
 AP-0000000C@ I*4   N

ARRAYS

    Address   Type  Name                Bytes  Dimensions

 AP-00000004@ R*8   A                     **   (*, *)

LABELS

    Address   Label

      **       10
```

```
C
        SUBROUTINE ROT( M, N, X, Y, Z, XROT, YROT, ZROT )
C
        IMPLICIT REAL*8 (A-H,O-Z)
        REAL*8 X(M,N), Y(M,N), Z(M,N), XROT, YROT, ZROT
C
C  THIS SUBROUTINE WILL ROTATE THE POINTS DESCRIBED
C  BY THE X , Y , AND Z ARRAYS. X IS THE DEPTH AWAY
C  FROM THE USER Z IS THE VERITCAL COMPONENT.
C
C *** CALCULATE THE SINES AND COSINES .
         RC= .01745329
         T = XROT * RC
         SXROT = DSIN( T )
         CXROT = DCOS( T )
         T = YROT * RC
         SYROT = DSIN( T )
         CYROT = DCOS( T )
         T = ZROT * RC
         SZROT = DSIN( T )
         CZROT = DCOS( T )
C **** CALCULATE CONSTANTS
        XC1 = CYROT*CZROT
        XC2 = SZROT*CYROT
        XC3 = SYROT
        YC1 = SXROT*SYROT*CZROT - CXROT*SZROT
        YC2 = SXROT*SYROT*SZROT + CXROT*CZROT
        YC3 = SXROT*CYROT
        ZC1 = CXROT*SYROT*CZROT + SXROT*SZROT
        ZC2 = CXROT*SYROT*SZROT - SXROT*CZROT
        ZC3 = CXROT*CYROT
C ***** NOW ROTATE IT.
        DO 100 I = 1 , M
        DO 100 J = 1 , N
        XT = X(I,J)
        YT = Y(I,J)
        ZT = Z(I,J)
        X(I,J) = XT*XC1 + YT*XC2 - ZT*XC3
        Y(I,J) = XT*YC1 + YT*YC2 + ZT*YC3
        Z(I,J) = XT*ZC1 + YT*ZC2 + ZT*ZC3
 100     CONTINUE
        RETURN
        END
```

```
ROT                                                  28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56            Page  34
                                                     28-May-1985 11:11:27    DSK0:[LAW.ROBOT]ROBOT.FOR;3

PROGRAM SECTIONS

    Name                                    Bytes   Attributes

  0 $CODE                                     504   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  2 $LOCAL                                    332   PIC CON REL LCL NOSHR NOEXE   RD   WRT QUAD

    Total Space Allocated                     836

ENTRY POINTS

    Address  Type  Name

  0-00000000       ROT

VARIABLES

    Address  Type  Name               Address  Type  Name               Address  Type  Name               Address  Type  Name

  2-00000018  R*8  CXROT          2-00000028  R*8  CYROT          2-00000038  R*8  CZROT          2-000000A0  I*4  I
  2-000000A4  I*4  J             AP-00000004@ I*4  M             AP-00000008@ I*4  N              2-00000000  R*8  RC
  2-00000010  R*8  SXROT          2-00000020  R*8  SYROT          2-00000030  R*8  SZROT          2-00000008  R*8  T
  2-00000040  R*8  XC1            2-00000048  R*8  XC2            2-00000050  R*8  XC3           AP-00000018@ R*8  XROT
  2-00000088  R*8  XT             2-00000058  R*8  YC1            2-00000060  R*8  YC2            2-00000068  R*8  YC3
 AP-0000001C@ R*8  YROT           2-00000090  R*8  YT             2-00000070  R*8  ZC1            2-00000078  R*8  ZC2
  2-00000080  R*8  ZC3           AP-00000020@ R*8  ZROT           2-00000098  R*8  ZT

ARRAYS

    Address  Type  Name                 Bytes  Dimensions

 AP-0000000C@ R*8  X                      **  (*, *)
 AP-00000010@ R*8  Y                      **  (*, *)
 AP-00000014@ R*8  Z                      **  (*, *)

LABELS

    Address   Label

      **      100

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name            Type  Name

   R*8  MTH$DCOS         R*8  MTH$DSIN
```

```
ROT                                                     28-May-1985 11:11:36    VAX-11 FORTRAN V3.4-56                  Page  35
                                                        28-May-1985 11:11:27    DSK0:[LAW.ROBOT]ROBOT.FOR;3

COMMAND QUALIFIERS

  FORTRAN /LIS ROBOT

  /CHECK=(NOBOUNDS,OVERFLOW,NOUNDERFLOW)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /STANDARD=(NOSYNTAX,NOSOURCE_FORM)
  /SHOW=(NOPREPROCESSOR,NOINCLUDE,MAP)
  /F77  /NOG_FLOATING  /I4  /OPTIMIZE  /WARNINGS  /NOD_LINES  /NOCROSS_REFERENCE  /NOMACHINE_CODE  /CONTINUATIONS=19

COMPILATION STATISTICS

  Run Time:             24.65 seconds
  Elapsed Time:         100.09 seconds
  Page Faults:          979
  Dynamic Memory:       154 pages
```

APPENDIX V.6   Benchmark Programs in Fortran and C.

Floating Point Operations.

```
      program float
      a=10.
      b=15.
      write(*,*) 'enter number of iterations  '
      read(*,*) numb
      do 10 i=1,numb
        x1=a+b
        x2=a*b
        x3=a/b
        x4=x1*x2*x3
10    continue
      stop
      end
```

```
main()
{   float a=10., b=15.;
    float x1,x2,x3,x4;
    int number,icount;
    printf("enter number of iterations  ");
    scanf("%d",&number);
    printf("%d",number);
    for(icount=1;icount<number;++icount)
    {  x1=a+b;
       x2=a*b;
       x3=a/b;
       x4=x1*x2*x3;
    }
}
```

Integer Operations.

```
      program fint
      integer a, b, x1,x2,x3,x4
      a=10
      b=15
      write(*,*) 'enter number of iterations  '
      read(*,*) numb
      do 10 i=1,numb
        x1=a+b
        x2=a*b
```

```
            x3=a/b
            x4=x1*x2*x3
 10      continue
         stop
         end
```

```
main()
{   int a=10, b=15;
    int x1,x2,x3,x4;
    int number,icount;
    printf("enter number of iterations  ");
    scanf("%d",&number);
    printf("%d",number);
    for(icount=1;icount<number;++icount)
    {  x1=a+b;
       x2=a*b;
       x3=a/b;
       x4=x1*x2*x3;
    }
}
```

Data Moves.

```
        program fdata
        real matrix(4,4),vector(4)
        write(*,*) 'enter number of iterations  '
        read(*,*) numb
        do 10 i=1,numb
        do 20 j=1,4
        do 30 k=1,4
           matrix(j,k)=vector(k)
30      continue
20      continue
10      continue
        stop
        end
```

```
main()
{
    int number,icount;
    int ind2,ind3;
    float matrix[4][4], vector[4];
    printf("enter number of iterations  ");
    scanf("%d",&number);
    for(icount=1;icount<number;++icount)
       for (ind2=0;ind2<4;++ind2)
```

```
          for (ind3=0;ind3<4;++ind3)
             matrix[ind3][ind2]=vector[ind3];
}
```

Nested Loops.

```
        program floop
        real matrix(4,4),vector(4)
        write(*,*) 'enter number of iterations  '
        read(*,*) numb
        do 10 i=1,numb
           do 20 j=1,4
              do 30 k=1,4
                 continue
30            continue
20         continue
10      continue
        stop
        end
```

```
main()
{
    int number,icount;
    int ind2,ind3;
    int itotal=0;
    float matrix[4][4], vector[4];
    printf("enter number of iterations  ");
    scanf("%d",&number);
    for(icount=1;icount<number;++icount)
       for (ind2=0;ind2<4;++ind2)
          for (ind3=0;ind3<4;++ind3)
             continue;
}
```

Single depth loop.

```
        program floop
        real matrix(4,4),vector(4)
        write(*,*) 'enter number of iterations  '
        read(*,*) numb
        do 10 i=1,numb
           continue
10      continue
        stop
```

```
      end


main()
{
     int number,icount;
     int ind2,ind3;
     int itotal=0;
     float matrix[4][4], vector[4];
     printf("enter number of iterations  ");
     scanf("%d",&number);
     for(icount=1;icount<number;++icount)
                continue;
}
```